Actuación en código máquina

Los programas en código máquina se ejecutan muy rápidamente, en concreto cientos de veces más rápido que sus equivalentes en BASIC.

Son tan potentes que permiten a su computadora hacer cosas ni soñadas por su fabricante.

El ZX Spectrum utiliza un microprocesador Z-80 y los programas, en código máquina, están escritos en lenguaje ensamblador para dicho microprocesador. Este libro es, de principio a fin, un curso completo en lenguaje ensamblador para el Z-80. Por ser su autor un profesor y programador experimentado, es ideal para cualquier persona con conocimientos en BASIC y contiene numerosos programas completos y muy útiles.

PUBLICACIONES SOBRE EL ZX SPECTRUM (TS 2068)

BISHOP: ZX SPECTRUM (TS 2068). Teoría y proyectos de Interfases.
HURLEY: ZX SPECTRUM (TS 2068). Introducción al procesamiento de textos.
LEWART: Programas de Ciencia e Ingeniería para microcomputadoras ZX-81 compatibles con el ZX Spectrum.
STREFA: ZX SPECTRUM (TS 2068). Técnicas de procesamiento de la información.
WILLIAMS: ZX SPECTRUM (TS 2068). Diseño y programación de juegos.

OTRAS OBRAS DE INTERES

PHILIPS: Programando el DRAGON juegos y gráficos.
GOSLING: Programación estructurada para microcomputadoras.
HURLEY: Introducción a la programación ZX-81 (TS 1000).

ISBN 968-451-337-0

ZX Spectrum (TS 2068) Programación en lenguaje ensamblador Tony Woods

ZX Spectrum (TS 2068)
Programación en lenguaje ensamblador

Tony Woods

ZX Spectrum (TS 2068)

PROGRAMACION EN LENGUAJE ENSAMBLADOR

ZX Spectrum (TS 2068)
Programación en lenguaje ensamblador

GRUPO-1

S.E.S.A.

Tony Woods

Traducción

Luis Joyanes Aguilar
Capitán de Artillería
Licenciado en Ciencias Físicas
Profesor de Electrónica Digital y Computadores
Academia de Artillería de Madrid

José Carlos Sastre Torres
Licenciado en Ciencias Físicas

Revisión técnica

Antonio Vaquero Sánchez
Catedrático de Informática
Facultad de Ciencias Físicas
Universidad Complutense

McGraw-Hill

MADRID • BOGOTÁ • BUENOS AIRES • GUATEMALA • LISBOA • MÉXICO • NUEVA YORK •
PANAMÁ • SAN JUAN • SANTIAGO • SAO PAULO
AUCKLAND • HAMBURGO • JOHANNESBURGO • LONDRES • MONTREAL • NUEVA DELHI •

ZX Spectrum. Programación en lenguaje ensamblador

ISBN: 968-451-727-0

Traducido de la primera edición en inglés de
LEARN AND USE ASSEMBLY LANGUAGE ON THE ZX SPECTRUM

ISBN: 0-07-084705-3

Edición exclusiva para Ediciones La Colina, S. A. (España)

ISBN: 84-7615-009-1
Depósito legal: M. 36.405-1984

Compuesto por FER Fotocomposición, S. A. - Lenguas, 8 - Madrid

Artes Gráficas EMA. Miguel Yuste, 27

PRINTED IN SPAIN-IMPRESO EN ESPAÑA

CONTENIDO

PREFACIO

Puesto que está leyendo esto, quizá se sienta poco satisfecho con lo que puede hacer su Spectrum trabajando en BASIC. Los programas en lenguaje ensamblador le permiten controlar directamente el número procesador, el cerebro de su Spectrum. Tiene un control total sobre todas las posibilidades incorporadas en su computadora.

Hay dos razones fundamentales para escribir programas en lenguaje ensamblador en vez de en BASIC; primero obtendrá una velocidad de procesamiento tremendamente superior, segundo sus programas ocuparán mucho menos memoria.

El incremento de velocidad es difícil de apreciar hasta que la vea. Le permitirá realizar un movimiento suave de sus gráficos a la velocidad que desee. Normalmente, el movimiento de gráficos necesitará detenerse para proporcionar una visualización satisfactoria. Bajo circunstancias normales un programa en lenguaje ensamblador será por lo menos veinte veces más rápido que el mismo programa en BASIC y puede llegar a ser hasta 200 veces más rápido.

La reducción de memoria utilizada por un programa en lenguaje ensamblador significa que puede tener más datos en memoria para que su programa los utilice. Una ventaja más del lenguaje ensamblador es que le proporciona un control completo en la forma de almacenar los datos. Mediante las técnicas avanzadas de almacenamiento puede, en ciertos casos, reducir el espacio requerido para almacenar los datos a una fracción del espacio requerido por BASIC.

Este libro enseña cómo escribir programas en lenguaje ensamblador. Antes de que se pueda ejecutar un programa en lenguaje ensamblador tiene que traducirse a código máquina, que es el único lenguaje que directamente comprende la computadora. Puesto que un programa en código máquina consiste en una serie de números, es irrazonable escribir un programa directamente en código máquina. El lenguaje ensamblador es el lenguaje más sencillo de comprender y el más cercano al código máquina.

La traducción de lenguaje ensamblador a lenguaje máquina es mejor que la realice la computadora mediante un programa llamado ensamblador. Puede llevarse a cabo de forma manual, pero excepto para pequeños programas es una tarea muy tediosa y sujeta a errores. Hay varios programas ensambladores para el Spectrum; todos los programas de este libro se han obtenido mediante el Ensamblador a Código Máquina de ZX Spectrum. Este programa tiene todas las facilidades necesarias para producir y traducir

programas en lenguaje ensamblador. Además es sencillo de utilizar. Es el ideal para el recién llegado al lenguaje ensamblador además de proporcionar las facilidades necesarias para el programador experimentado en lenguaje ensamblador.

El lenguaje ensamblador es sencillamente otro lenguaje de programación y comprobará que no es mucho más difícil que el BASIC. Como con el resto de los lenguajes de programación la única manera de aprenderlo es escribiendo muchos programas. Al final de cada capítulo ya ha supuesto un problema de programación para que lo intente el lector. Cada programa está diseñado considerando la capacidad del lector y exigiendo un poco de razonamiento.

Finalmente, quiero dar las gracias a mi esposa, Marilyn, por escribir a máquina el texto y por soportarme durante su escritura. Quiero dar también las gracias a mi hijo, Richard, por ayudarme a introducir y depurar los programas ejemplo.

TONY WOODS

1 INTERIORIDADES DEL SPECTRUM

1.1 La computadora Spectrum

El Spectrum es una potente computadora personal. Puede utilizarse para una gran variedad de aplicaciones; puede entretener, enseñar, incluso puede utilizarse para realizar tareas de administración.

La mayoría de las aplicaciones de la computadora incluyen tres etapas independientes:

1. Colocar información en la computadora.
2. Procesamiento de esa información.
3. Visualización de los resultados del procesamiento.

La figura 1.1 muestra un sistema Spectrum típico. La mayoría de la información se introduce en la computadora mediante el te-

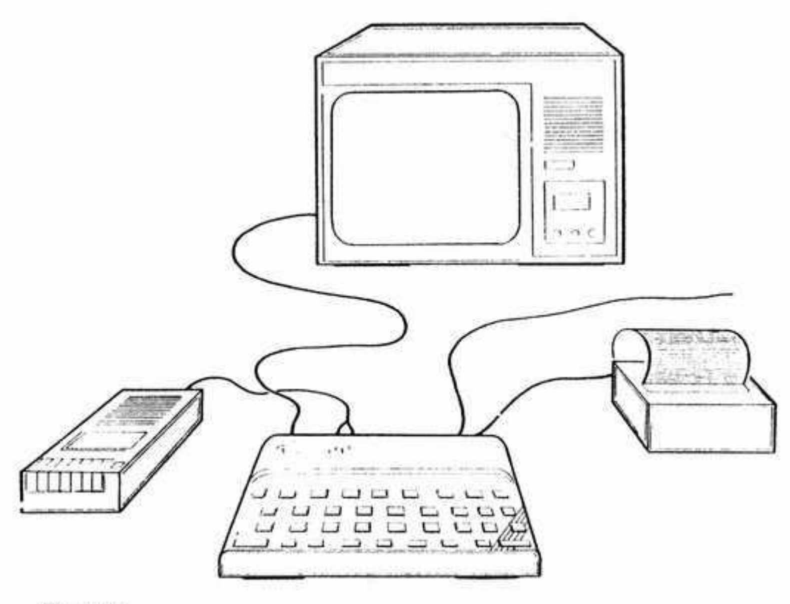


Figura 1.1

clado, pero también procede de otras fuentes tales como los controles de juegos, sensores de temperatura o incluso la información que se ha almacenado previamente en una cinta de casete o microdrive (microunidad) de cartuchos. Cuando la información entra en la computadora se almacena o conserva en la memoria interna de la computadora.

La siguiente etapa, después de la entrada de información, es el procesamiento de la misma para producir los resultados requeridos. Esta se realiza por una parte de la computadora conocida como unidad central de procesamiento. La unidad central de procesamiento del Spectrum está incorporada en un solo chip de silicona que se le llama microprocesador. El que se utiliza en el Spectrum es un Z80A. Es el mismo microprocesador que se utiliza en otras computadoras de administración de alto precio.

Finalmente, se tienen que sacar de la computadora los resultados del procesamiento. Generalmente estos resultados se sacan a una pantalla de televisión, pero también se pueden entregar en forma de sonido en un altavoz, en forma de impreso mediante una impresora o conservar la información en una cinta de casete o microdrive, e incluso en forma de señales para controlar dispositivos externos, tales como las válvulas de un sistema de calefacción central.

1.2 Lenguajes de la computadora

Antes de que se pueda utilizar una computadora para cualquier aplicación, se le tienen que dar instrucciones que indiquen exactamente lo que ha de hacer. A estas instrucciones se las llama *programa.*

Hay muchos tipos diferentes de lenguajes de programación, la mayoría de los cuales están diseñados para que se adapten a un tipo determinado de aplicación para la computadora. Por ejemplo, la figura 1.2 muestra un pequeño programa escrito en el lenguaje COBOL. Este lenguaje está pensado para utilizarlo en programas de administración y su ventaja principal es que es muy sencillo de leer para las personas que no son programadores.

Otro lenguaje de programación que le resultará familiar es el BASIC. El BASIC fue diseñado como lenguaje de propósito general que fuese fácil de aprender y utilizar.

El COBOL y el BASIC son *lenguajes de alto nivel.* Esto quiere decir que fueron diseñados para satisfacer un tipo particular de utilización y los programas producidos con estos lenguajes se podrían ejecutar en un amplio espectro de computadoras. Puesto que no están relacionados directamente con una computadora en


```
IDENTIFICATION DIVISION.
PROGRAM-ID.EJEMPLO.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
  SELECT FICHERO-F ASSIGN TO DISK.
  SELECT IMPRESO ASSIGN TO $LPT.
DATA-DIVISION.
FD FICHERO-F.
DATA RECORD IS REGISTRO-F.
01 REGISTRO-F.
  02 DPT          PICTURE 99999.
  02 CANTIDAD     PICTURE 99999V99.
FD IMPRESO.
DATA RECORD IS LINEA.
01 LINEA.
  02 TEXTO     PICTURE X(15).
  02 PDPTO     PICTURE 99999.
  02 PSUMA     PICTURE 99999.99.
WORKING-STORAGE SECTION.
77 ACTUAL      PICTURE 99999.
77 SUMA        PICTURE 99999V99.
PROCEDURE DIVISION.
  MOVE 'TOTAL DEL DPTO.'TO TEXTO.
  OPEN INFUT FICHERO-F.
  OPEN OUTPUT LINEA.
  REAL FICHERO-F.
INIC.
  MOVE DPTO TO ACTUAL.
  MOVE CANTIDAD TO SUMA.
BUCLE.
  READ FICHERO-F.
  IF DPTO NOT EQUAL TO ACTUAL GOTO IMPRIMIR.
  ADD CANTIDAD TO SUMA.
  GOTO BUCLE.
IMPRIMIR.
  MOVE SUMA TO PSUMA.
  MOVE TOTAL TO PDPTO.
  WRITE LINEA.
  IF DPTO NOT EQUAL TO '99999' GOTO INIC.
  CLOSE FICHERO-F.
  CLOSE LINEA.
```

Figura 1.2

particular, no aprovechan todas las posibilidades incorporadas en la unidad central de procesamiento de la computadora; en el caso del Spectrum esto quiere decir que no utilizan plenamente todas las posibilidades del microprocesador Z80.

Este libro le dice cómo programar en el lenguaje ensamblador del Z80. Este lenguaje de programación está específicamente diseñado para el microprocesador Z80 y utiliza todas las posibilidades del Z80. Debido a que está diseñado para un tipo específico de unidad central de procesamiento (o microprocesador) se le conoce como un *lenguaje de programación de bajo nivel.*

1.3 El microprocesador Z80

Todas las microcomputadoras utilizan una unidad central de procesamiento que está contenida en un único chip de silicio. A este tipo de unidad central de procesamiento se le conoce como microprocesador. Hay muchos tipos de microprocesadores, cada uno con su propio lenguaje ensamblador. En este libro solamente estamos interesados en el lenguaje ensamblador del Z80 que es el que utiliza el Spectrum. Puesto que utilizaremos directamente todas las posibilidades del microprocesador, tenemos que analizar la estructura interna de esta unidad central de procesamiento.

La figura 1.3 muestra un esquema general del microprocesador. Básicamente consiste en un diferente número de secciones que van interconectadas mediante un bus (canal) de datos de ocho bits. Esto quiere decir que se pueden mover y procesar números binarios de ocho dígitos mediante el microprocesador.

Los componentes más importantes del Z80 para el programador son los registros. La figura 1.4 muestra un esquema de los registros del Z80. Un registro es un área (zona) de memoria que es capaz de contener un solo elemento de información. Los registros se utilizan para almacenar datos de forma temporal mientras se procesan o esperan a ser procesados.

Toda la información o los datos se almacenan y se utilizan dentro de la computadora como números binarios. Para aquellos que no estén familiarizados con los números binarios, se explicarán en el próximo capítulo. Debido a que los registros son las áreas de almacenamiento de la computadora, contienen los datos como números binarios. El microprocesador Z80 tiene algunos registros que pueden contener números binarios de ocho cifras o dígitos y otros que pueden contener números binarios de 16 dígitos. Cuando se habla sobre números binarios se utiliza el término *bit* como una abreviatura de las palabras inglesas «binary digits» o dígitos binarios. A los registros que contienen números binarios de ocho dígitos se les llama registros de ocho bits y a los que contienen números binarios de 16 dígitos se les llama registros de 16 bits.

El Z80 es un microprocesador de ocho bits lo cual significa

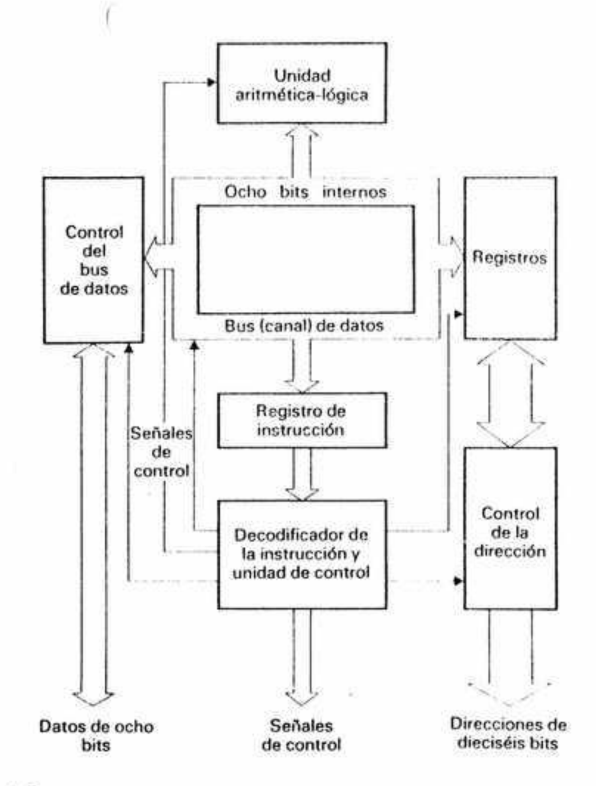


Figura 1.3

que la mayoría de los datos utilizados consisten en números de ocho bits. Debido a que el Z80 es un microprocesador avanzado de ocho bits, incluye también algunas facilidades para procesar números de 16 bits.

1.4 Los registros

El registro más importante es el registro A, también llamado acumulador. Es un registro de ocho bits y se utiliza para la mayoría de las operaciones aritméticas y para otro tipo de procesamiento de operaciones como las comparaciones. Por ejemplo, cuando se suman dos números se pone uno de ellos primero en el registro A, después se le suma el segundo número y el resultado se deja en el registro A.

A	F	A'	F'
B	C	B'	C'
D	E	D'	E'
H	L	H'	L'
IX			
IY			
SP			
PC			
I			
R			

Figura 1.4

El registro F, llamado registro indicador o de estado (flag), es muy especial. Se utiliza para indicar que se han producido diferentes condiciones debido al procesamiento que ha tenido lugar. Como ejemplo, después de realizar una operación, como la simple suma del párrafo anterior, alguna parte del registro F nos indicará si el resultado es positivo o negativo o sí es cero o no. Esto se puede comprobar después mediante instrucciones en el programa, similares a la sentencia IF de BASIC. El registro F no lo utiliza el programador de forma directa.

Los registros B, C, D, E, H y L son todos registros de ocho bits de propósito general; se pueden utilizar por parejas, llamados BC, DE y HL, como registros de 16 bits. Aunque son de propósito general y pueden virtualmente utilizarse indistintamente, se tiende de forma convencional a utilizarlos en tareas específicas. Su uso se indicará en el momento adecuado; no obstante, y como ejemplos, la pareja de registros HL se utiliza con frecuencia como apuntador (puntero) para señalar un lugar específico de la memoria de la computadora.

El registro PC es un registro de 16 bits cuyo propósito es indicar a la computadora dónde puede encontrar en memoria la siguiente instrucción del programa.

El registro SC es un registro de 16 bits que se utiliza para realizar (implementar) una pila (stack). Es un dispositivo de programación muy útil que explicaremos más adelante.

Los dos registros IX e IY son registros de 16 bits y se llaman registros índices. Se utilizan cuando el programador desea utilizar tablas o listas de datos. El registro IY no debería utilizarlo el programador de un Spectrum, porque lo utiliza el sistema del Spectrum.

Finalmente hay dos registros de ocho bits, el I y el R, que tienen una utilización muy especial en las ejecuciones de la computadora; es muy raro que los utilice el programador.

2 BITS Y BYTES

2.1 Números binarios

Dentro de la computadora todos los datos están formados por grupos de pulsos eléctricos. Para mayor sencillez a la hora de escribir decimos que la presencia de un pulso eléctrico puede representarse por el dígito 1 y la ausencia de pulso puede representarse por el dígito 0. Esto quiere decir que todos los datos en la computadora pueden representarse por números formados exclusivamente por ceros y unos. Los números que solamente utilizan ceros y unos se llaman *números binarios.*

Antes de analizar los números binarios es más práctico reconsiderar algunas ideas que utilizamos con los números decimales cada día. Por ejemplo, el número 764 utiliza los tres dígitos 7, 6 y 4, pero puesto que están en el orden indicado, sabemos que el siete representa siete centenas, el seis, seis decenas y el cuatro, cuatro unidades. Podríamos escribir:

$$764 = 7 \times 100 + 6 \times 10 + 4 \times 1$$

o como:

$$764 = 7 \times (10 \times 10) + 6 \times (10) + 4 \times (1)$$

Esto quiere decir que cada posición a la izquierda es diez veces más que la posición anterior. Los números decimales utilizan el factor 10 para cada posición, otra forma de llamar a los números decimales es números en base 10.

Aunque se utilice el 10 como base para los números decimales se puede utilizar cualquier número como base. Las bases que con más frecuencia se utilizan en las computadoras son la base 2, para los números binarios, y la base 16, conocida como números hexadecimales. Los números binarios son los más importantes para las computadoras puesto que todos los datos en la computadora están en forma de números binarios. Los números hexadecimales se utilizan porque proporcionan una forma sencilla y rápida de escribir números binarios.

Los números binarios y hexadecimales pueden desglosarse en dígitos separados de la misma forma que un número decimal. Podemos escribir el número binario 1011B como:

$$1011B = 1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1$$

o como:

$$1011B = 1 \times (2 \times 2 \times 2) + 0 \times (2 \times 2) + 1 \times (2) + 1 \times (1)$$

y el número hexadecimal 945H como:

$$945H = 9 \times 256 + 4 \times 16 + 5 \times 1$$

o como:

$$945H = 9 \times (16 \times 16) + 4 \times (16) + 5 \times (1)$$

La letra B al final del número binario se utiliza para indicar que el número es binario y no en cualquier otra base de numeración. De la misma forma, la H al final del número hexadecimal se utiliza para indicar que el número es hexadecimal (el ensamblador del Spectrum utiliza el convenio de colocar un signo dólar en la parte izquierda de un número para indicar que es hexadecimal; por ejemplo, $1234 es lo mismo que 1234H).

Cualquier número decimal puede escribirse utilizando los dígitos del 0 al 9, es decir, utilizando los dígitos del cero hasta uno menos que el valor de la base. Para cualquier base de numeración necesitamos símbolos para los números que vayan de cero a uno menos que el valor de su base.

Con los números binarios, mediante los dígitos 0 y 1, podemos escribir cualquier número binario. Los números hexadecimales necesitan símbolos para los dígitos del 0 hasta un valor de 15. Para los valores del 0 al 9 utilizamos los mismos símbolos que para los números decimales, es decir, los dígitos del 0 al 9, pero para los valores del 10 al 15 no podemos utilizar los mismos que

Decimal	Binario	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Figura 2.1

los números decimales porque sería muy confuso; utilizando las letras de la A a la F para representar los valores del 10 al 15. La figura 2.1 muestra la equivalencia binaria, hexadecimal y decimal para los números hasta el 15.

2.2 Cambio de base de numeración

De binario y hexadecimal se puede convertir a decimal escribiendo en forma desarrollada, como mostrábamos al comienzo de esta sección, y realizando los cálculos. La figura 2.2 muestra unos ejemplos de conversión de un número binario y otro hexadecimal.

```
01011011B = 1 x 1 + 1 x 2 + 0 x 4 + 1 x 8 + 1 x 16 + 0 x 32 + 1
              x 64 + 0 x 128
          = 1 + 2 + 0 + 8 + 16 + 0 + 64 + 0
          = 91
    5AC7H = 7 x 1 + C x 16 + A x 256 + 5 x 4096
          = 7 x 1 + 12 x 16 + 10 x 256 + 5 x 4096
          = 7 + 192 + 2560 + 20480
          = 23239
```

Figura 2.2

La conversión de un número decimal a binario es una tarea ligeramente más compleja; supone la división continuada del decimal entre 2 y la anotación del resto en cada etapa. La figura 2.3 muestra este proceso para el número decimal 245. Observe que el número binario lo puede encontrar leyendo los restos de abajo hacia arriba.

```
2)245(1
2)122(0
2)61(1
2)30(0
2)15(1
2)7(1
2)3(1
2)1(1
   0
```

Figura 2.3

La conversión de decimal a hexadecimal es similar a la conversión de decimal a binario excepto que ahora supone una división continua entre 16.

Los números pueden convertirse de binario a hexadecimal dividiendo el número binario en grupos de cuatro bits, comenzando por el lado derecho del número, y añadiendo ceros extra si es necesario por el extremo izquierdo para formar el último grupo de cuatro bits. Después se convierte cada grupo de cuatro bits a su equivalente hexadecimal como se muestra en la figura 2.1.

La conversión de hexadecimal a binario se lleva a cabo reemplazando cada dígito hexadecimal por su grupo de cuatro bits equivalente tomados de la figura 2.1. La figura 2.4 muestra ejemplos de estas conversiones.

2.3 Bits y bytes

La unidad básica de datos en el Spectrum se llama *byte*; un byte es un número binario de ocho bits. El valor de un byte puede utilizarse para representar diferentes cosas, tales como números, caracteres, o incluso instrucciones de un programa. Es importante comprender que el mismo número binario puede representar una de estas cosas dependiendo de su posición en memoria. La representación de caracteres se tratará en el Capítulo 5.

La representación numérica indica que el contenido del byte se considera como un número binario. Por tanto, un byte que contenga el dígito binario 01101101 representa al número 1101101B, que es el 109 en decimal. El rango de números que puede contenerse en un solo byte va desde el 00000000B al 11111111B, que en decimal es del 0 al 255. Este método sólo se puede utilizar para representar números positivos; se conoce como representación de números sin signo para distinguirlo del siguiente método, que se utiliza para representar números positivos y negativos.

Con frecuencia necesitaremos utilizar números que tengan valor positivo o negativo. Para ellos se utiliza otro tipo de representación, se la llama «complemento a dos» o números con signo. La base de este método es que en la computadora el número de dígitos en forma binaria de cualquier número es siempre el mismo. El número de dígitos le determina usted, el programador,

```
0011 1011 0111 1010
  3    B    7    A
0011101101111010B = 3B7AH

  5    A    C    7
0101 1010 1100 0111
5AC7H = 0101101011000111B
```

Figura 2.4

para dar un rango suficiente al problema. El número de dígitos tiene que ser un múltiplo de ocho y usualmente el número de dígitos binarios es ocho, que por supuesto es un byte.

Mediante este método cada posición de bit en el número tiene su valor usual excepto el del bit de extremo izquierdo que tiene el valor usual pero en negativo. La figura 2.5 muestra el valor de cada bit en un número de ocho bits para los números sin signo y con signo.

En el número con signo la posición del dígito de la izquierda representa el valor −128 y el resto de los dígitos representan su valor normal de posición. Mediante estos valores, un número positivo solamente utilizará siete posiciones y el dígito de la izquierda siempre será cero; los valores negativos se obtendrán sumando la cantidad necesaria a −128. La figura 2.5 presenta algunos ejemplos de números con signos positivos y negativos.

La computadora no utiliza este método de buscar «el complemento a dos de los números». Para encontrar el número negativo partiendo del positivo, utiliza el simple proceso de tomar la representación binaria del número positivo y convertir todos los

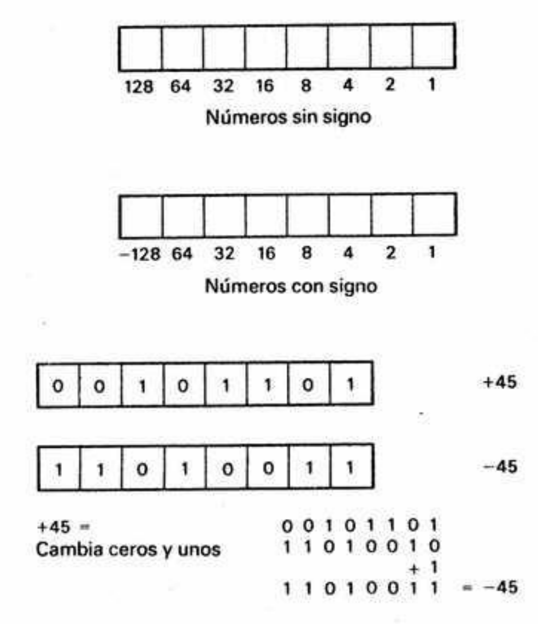


Figura 2.5

ceros en unos y todos los unos en ceros, y finalmente sumando uno a este nuevo número. También se muestra esto en la figura 2.5.

2.4 Memoria

La memoria del Spectrum puede imaginarse como un determinado número de cajas, llamadas posiciones, las cuales pueden contener un número binario de ocho bits. Puesto que cada posición contiene ocho bits es conveniente referirse a las posiciones como bytes de memoria, aunque de forma estricta sólo se debería utilizar para referirse al contenido de una posición de memoria.

Los bytes de la memoria van numerados en secuencia comenzando desde cero. Al número ubicado en cada byte se le atribuye una dirección exactamente de la misma forma que las casas dentro de una calle van numeradas para dar una dirección. Estos números se utilizan de la misma forma para encontrar un byte determinado en memoria.

En la programación en lenguaje ensamblador, el programador decide qué posiciones de memoria utilizará para almacenar los datos. A menudo el programador puede querer utilizar los datos que están almacenados en una determinada posición de memoria. Una forma abreviada de escribir en lenguaje ensamblador «el valor del byte de memoria cuya dirección es», es encerrar la dirección entre paréntesis, así que, por ejemplo, (23 637) no asigna el valor 23 637, sino el valor del byte de memoria cuya dirección es 23 637.

Dentro del Spectrum hay dos tipos diferentes de memoria, llamadas ROM y RAM. La ROM, que viene de «Read Only Memory» (Memoria de sólo lectura), es la memoria que contiene permanentemente un programa; no puede modificarla el programador, aunque puede utilizar el programa o parte del programa en sus propios programas. La memoria ROM del Spectrum utiliza las posiciones de memoria que van desde la dirección 0 a la dirección 16 383. El resto de la memoria es la RAM, que viene de «Random Access Memory» (Memoria de acceso aleatorio). Esta memoria puede modificarla el programador, aunque hay algunas posiciones que no es recomendable variar. No se puede producir un deterioro real por modificar cualquier byte de la RAM, aunque en algunos casos puede provocar una «caída» de la computadora. Esto no es tan serio como suena; en el peor de los casos puede significar que tenga que apagar y encender de nuevo la computadora para recuperar el control.

La máquina de 16K estándar tiene 16 veces 1024 posiciones

de memoria RAM. (La letra K en la jerga de computadoras representa al número 1024.) La memoria RAM ocupa las posiciones que van desde la dirección 16 384 a la dirección 32 767. El Spectrum de 48K tiene 48K de memoria RAM que ocupan las posiciones de las direcciones 16 384 a la dirección 65 535. Esta es la dirección superior de memoria que puede utilizarse en el Spectrum sin utilizar técnicas especiales para dar a la memoria una única dirección.

3 PROGRAMACION EN LENGUAJE ENSAMBLADOR

3.1 Lenguaje ensamblador

Antes de que se pueda ejecutar cualquier programa en una computadora, se tienen que traducir todas las instrucciones en una serie de números binarios. La unidad central de procesamiento de la computadora, en nuestro caso el microprocesador Z80, solamente entiende las instrucciones de un programa en forma de números binarios.

La principal función de la memoria ROM, en el Spectrum, es la conversión de instrucciones BASIC en instrucciones en forma de números binarios, conocidas como código máquina. Puesto que el BASIC no fue escrito específicamente para el microprocesador Z80, la conversión a código máquina es relativamente ineficaz y en términos de computadoras es muy lenta.

Aunque no es posible escribir un programa e introducirlo en el Spectrum directamente en binario, es factible convertir los números binarios a decimal y utilizar la sentencia POKE para poner la instrucción en memoria. Es fácil imaginar que una de las acciones de la sentencia POKE es convertir el número decimal a binario antes de ponerlo en memoria.

La figura 3.1 muestra un pequeño programa en este formato decimal; como puede observar, la lectura de este programa no le da ninguna idea de cómo funciona ninguna de las instrucciones. El introducir programas de esta forma es muy tedioso y una fuen-

Dirección	Contenido
23760	50
23761	0
23762	120
23763	33
23764	208
23765	92
23766	70
23767	128
23768	50
23769	209
23770	92
23771	201

Figura 3.1

te de errores asegurada y éstos son muy difíciles de encontrar. Recuerde que éste es solamente un programa muy corto, por lo que puede imaginarse los problemas que pueden aparecer con programas más largos.

Para utilizar todas las posibilidades del microprocesador Z80, necesita un lenguaje que no solamente esté profundamente relacionado con el código máquina utilizado por la computadora, sino también que sea fácil de leer y comprender. El lenguaje que cumple estos requisitos es el *lenguaje ensamblador*.

La pretensión principal del lenguaje ensamblador es hacer comprensible el código máquina de la computadora. Por ejemplo, la primera instrucción de la figura 3.1, el decimal 120, se convierte en el número binario 01111000B. Esto representa la instrucción de cargar el contenido del registro B en el acumulador. En el lenguaje ensamblador esta instrucción se representaría por:

LD A,B

Eche un vistazo rápido al Apéndice A que proporciona una lista completa de todas las instrucciones que reconoce el microprocesador Z80; no se preocupe si no comprende muchas de ellas, porque las explicaremos posteriormente. La figura 3.2 muestra el programa de la figura 3.1 escrito en lenguaje ensamblador. Debería observar que aunque el número 120 es la primera instrucción en el programa en código máquina, su equivalente en el lenguaje

```
10 REM go
12 REM org 23760
13 REM !reservar espacio para
   números
16 REM Num2;defb 50
18 REM Result;defb 0
19 REM !cargar primer número
20 REM ld a,b
21 REM !cargar segundo número
22 REM ld hl, Num2
24 REM ld b,(hl)
25 REM !realice la suma
26 REM add a,b
27 REM !guarde el resultado
28 REM ld (Result),a
29 REM !retorno a BASIC
30 REM ret
32 REM finish
```

Figura 3.2

ensamblador, LD A,B, no es la primera instrucción del programa en lenguaje ensamblador. Va precedida por otras instrucciones que pasan información al programa que traduce este programa del lenguaje ensamblador al código máquina.

Los lenguajes ensambladores utilizan pequeños grupos de letras para indicar la operación que llevan a cabo. Las letras se eligen para ayudar al programador a recordar qué operación va a utilizar. Por ejemplo, cuando se cargan datos en un registro desde otro registro se utilizan las letras LD (del inglés Load «cargar») y cuando se va a sumar se utilizan las letras ADD (del inglés Add «sumar»). Hay muchas más facilidades en el lenguaje ensamblador; las veremos más tarde.

Cuando se escriben instrucciones en lenguaje ensamblador, la posición dentro de la instrucción de los espacios en blanco, las comas o los paréntesis es crítica. Por ejemplo, la instrucción LD (32500), A la aceptaría un ensamblador como el ensamblador del ZX Spectrum, pero no ocurriría lo mismo con LD(32500),A o con LD (32500)A. Si cuando está ensamblando un programa recibe un mensaje de error, merece la pena realizar primero un análisis minucioso del formato de la instrucción.

3.2 Un programa ejemplo

Ahora es el momento de ver un programa en lenguaje ensamblador completo y analizar cómo puede utilizarse en el Spectrum. El programa que se muestra en la figura 3.3 es un programa de renumeración muy sencillo. Solamente vuelve a numerar los números de línea iniciales, no lo hace con los números de línea de las sentencias GOTO o GOSUB. En una fase posterior quizá quiera intentar una rutina más extensa. La mayoría de los programas en lenguaje ensamblador utilizados en el Spectrum se ejecutarán acompañando a un programa BASIC. El programa BASIC que acompaña al programa de renumeración se muestra en la figura 3.4.

```
10 REM go
20 REM org 23760 32500
22 REM !Programa para renumerar
   un programa BASIC
23 REM·!
25 REM !buscar el comienzo del
   programa
30 REM ld hl,23635
35 REM !poner DE a cero
```

```
 40 REM ld de,0
 50 REM Bucle;ex de,hl
 53 REM !Poner incremento entre
    líneas
 55 REM ld bc,10
 58 REM !calcular el número
    de línea
 60 REM add hl,bc
 70 REM ex de,hl
 75 REM !almacenar el número de línea
 80 REM ld (hl),d
 90 REM inc hl
100 REM ld (hl),e
110 REM inc hl
115 REM !buscar longitud de la
    línea
120 REM ld c,(hl)
130 REM inc hl
140 REM ld b,(hl)
150 REM dec hl
160 REM add hl,bc
165 REM !buscar número de la
    siguiente línea
170 REM ld d,(hl)
180 REM inc hl
190 REM ld e,(hl)
200 REM dec hl
205 REM !verificar final del
    programa buscando la línea 9000
210 REM ld a,d
220 REM sub 35
230 REM jp m,Bucle
240 REM ld a,e
250 REM sub 40
260 REM jp m,Bucle
270 REM ret
280 REM finish
```

Figura 3.3

```
9000 CLEAR 32500
9010 LOAD "figura 3.3."CODE
9020 RANDOMIZE USR 32500
9030 LIST
```

Figura 3.4

En este momento quizá no comprenderá la mayoría de las instrucciones del programa en lenguaje ensamblador aunque a lo mejor hay alguna que sí. Ahora tiene más interés el programa BASIC que le acompaña, que como puede ver es muy corto. La primera línea establece la variable del sistema, RAMTOP, para que deje parte de la memoria protegida para el programa en código máquina. La siguiente línea se utiliza para cargar el programa en código máquina en esta memoria protegida.

La línea siguiente es la más importante del programa; es la instrucción que produce la ejecución del programa en código máquina. El efecto de esta instrucción es que ejecuta el programa en código máquina como una subrutina del programa BASIC. Si no sabe cómo funcionan las subrutinas no se preocupe porque lo veremos más adelante en otra sección. La última instrucción lista el programa renumerado para mostrar que ha funcionado.

3.3 Instrucciones

Generalmente todas las instrucciones en código máquina utilizadas por el microprocesador constan de dos partes. La primera, llamada operación, indica a la computadora la acción a tomar, mientras que la segunda, llamada operando, indica a la computadora qué datos ha de utilizar. La primera instrucción de la figura 3.1 es el número decimal 120, que se convierte en el número binario 01111000. Los cinco primeros dígitos de este número, 011111, representan el código de la operación «cargar en el acumulador» y los tres dígitos finales, 000, representan el operando, en este caso los datos contenidos en el registro B. La instrucción completa es «cargar en el acumulador el valor del registro B».

Un nuevo vistazo al Apéndice A le mostrará que no todas las instrucciones son de la misma longitud; algunas sólo utilizan un byte, otras dos, otras tres y algunas cuatro. Estas diferencias son debidas principalmente a las distintas formas de especificar los operandos; a los que se conoce como modos de direccionamiento y los detallaremos más adelante.

La figura 3.5 muestra varias instrucciones de diferentes longitudes en lenguaje ensamblador y en código máquina.

3.4 Ensamblaje

Un programa que se haya escrito en lenguaje ensamblador tiene que traducirse a código máquina antes de que pueda ejecutarlo la computadora. Al contrario que los programas BASIC que se

Lenguaje ensamblador	Código máquina	
LD A,B	01111000B	120
LDIR	11101101B	237
	10110000B	176
JP 32000	11000011B	195
	00000000B	0
	01111101B	125
LD (32000),IX	11011101B	221
	00100010B	34
	00000000B	0
	01111101B	125

Figura 3.5

traducen a código máquina y se ejecutan línea a línea, el programa en lenguaje ensamblador se traduce por completo a código máquina antes de que se ejecute el programa. Esto quiere decir que se tiene que utilizar un área (zona) de memoria para almacenar el programa en código máquina además de la memoria utilizada para contener el programa en lenguaje ensamblador. La figura 3.6 muestra una utilización típica de la memoria con un programa en ensamblador.

La forma más conveniente de llevar a cabo la traducción es la utilización de un programa conocido como ensamblador; éste realizará la traducción al código máquina, comprobará errores del programa y cargará el programa a memoria. Todos los programas de este libro han sido ensamblados mediante el Ensamblador en Código Máquina del ZX Spectrum. Este programa amplía realmente su Spectrum. El Apéndice B describe la utilización de este ensamblador.

Si no dispone de un ensamblador, los programas pueden traducirse manualmente; a este proceso se le conoce como ensamblaje (o ensamblado) manual y se describe con detalle en el Apéndice D.

Cuando se ha traducido el programa a código máquina, se puede ejecutar en la computadora y se pueden detectar los errores y corregirlos. Comparándolo con los programas BASIC, este es un proceso más dificultoso porque los programas en código máquina erróneos no producen mensajes de error. Otro problema importante de los programas en código máquina es que la tecla BREAK no tiene ningún efecto, al menos que se escriba específicamente en el programa. Posteriormente se describirá un método para hacerlo. Si un programa en código máquina se introduce en un bucle infinito —generalmente cuando parece que la computadora no hace nada—, la única forma de recuperar el control es apagar y encender de nuevo la computadora. Verá que es una buena idea seguir la norma de guardar los programas antes de intentar ejecutarlos. Si después tiene la mala fortuna de introducirse en un bucle infinito no tendrá que volver a introducir todo su programa.

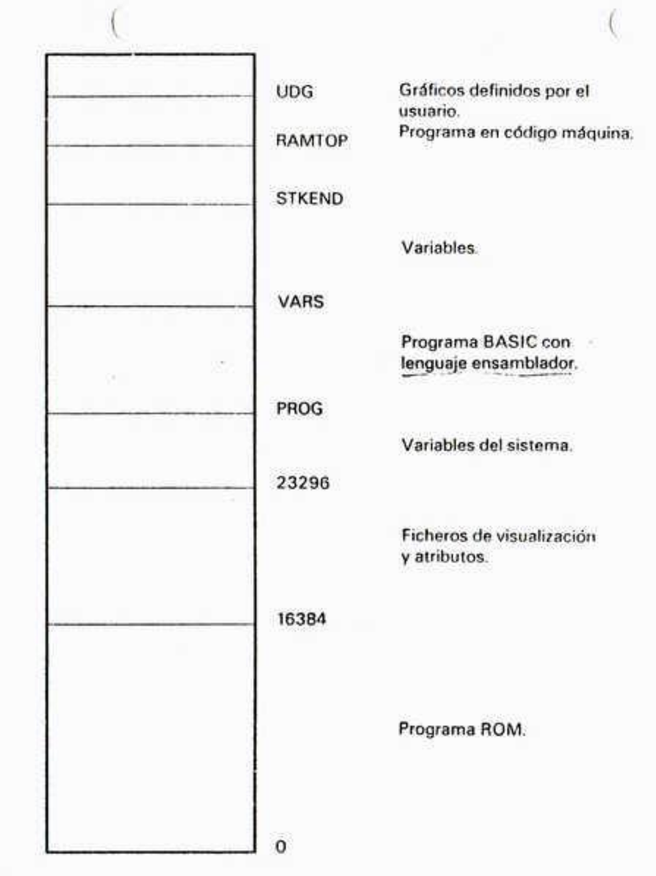


Figura 3.6

Puesto que el Spectrum se ha diseñado principalmente para ejecutar programas BASIC, todos los programas en código máquina se ejecutan como subrutinas para el sistema BASIC. La forma usual de comenzar un programa en código máquina es mediante la sentencia u orden RANDOMIZE USR XXXX, donde XXXX

es la posición de comienzo en memoria del programa en código máquina. La función USR es un tipo especial de llamada a una subrutina; se utiliza para una subrutina en código máquina de la misma forma que se utiliza el GOSUB para una subrutina en BASIC. Como todos los programas en código máquina se ejecutan como subrutinas para el sistema BASIC, todos tienen que terminar con la instrucción RET, que significa retorno de una subrutina, por lo que se devuelve control al sistema BASIC.

3.5 Almacenamiento de un programa

El programador de lenguaje ensamblador trabaja mucho más próximo al microprocesador de la computadora que aquel que utiliza el BASIC. Esto proporciona al programador un mayor control y una mejor utilización de las posibilidades que tiene el microprocesador; sin embargo, esto también quiere decir que él o ella va a tener menos privilegios y un determinado número de tareas que las realiza el BASIC de forma automática tendrá que realizarlas el programador. Una de estas tareas es decidir en qué dirección de la memoria de la computadora se deberá almacenar el programa en código máquina para que pueda ejecutarse fácilmente.

Esencialmente hay tres áreas o zonas diferentes de memoria que pueden utilizarse, cada una con sus ventajas y desventajas. El lugar más seguro para almacenar sus programas es la parte superior de la memoria. Se puede cargar el programa en la parte superior de la memoria y poner la variable del sistema RAMTOP con la posición de memoria justamente por debajo del comienzo del programa; así queda protegido el programa de que sea sobre escrito por el BASIC. La principal desventaja de esta parte de memoria es que el programa en código máquina y el programa BASIC que le acompaña, si es que lo hay, tienen que guardarse y cargarse por separado.

Si ha utilizado el código máquina en la computadora ZX81 probablemente habrá almacenado sus programas en una sentencia REM al principio de un programa BASIC. Esto se puede hacer también en el Spectrum y tiene la ventaja de que se guarda automáticamente el programa en código máquina con el programa BASIC. La desventaja de este método es que el comienzo de un programa BASIC no tiene una posición fija en el Spectrum, sino que depende de los dispositivos conectados al Spectrum. El comienzo del área del programa se puede localizar en la variable del sistema PROG.

Finalmente, el programa puede almacenarse en el área de memoria comprendida entre la pila del calculador y la pila de la máquina. Esta área está señalada por la variable del sistema STKEND.

Probablemente la mejor forma de trabajar es utilizar una sentencia REM para almacenar su programa mientras lo está desarrollando y haciendo que funcione, pero cuando ya esté depurado debería almacenarlo en la parte superior de la memoria y protegerlo del BASIC volviendo a establecer la variable RAMTOP mediante la sentencia CLEAR.

La figura 3.7 muestra estas tres áreas o zonas de la memoria.

3.6 Subrutinas

Las subrutinas son una técnica muy importante en la programación, especialmente para el programador en lenguaje ensamblador. Debido a su importancia, esta sección mostrará por qué se utilizan y cómo utilizarlas. No detallará como funcionan, ya que se verá en un capítulo posterior.

¿Por qué debería utilizar las subrutinas? Suponga que escribe

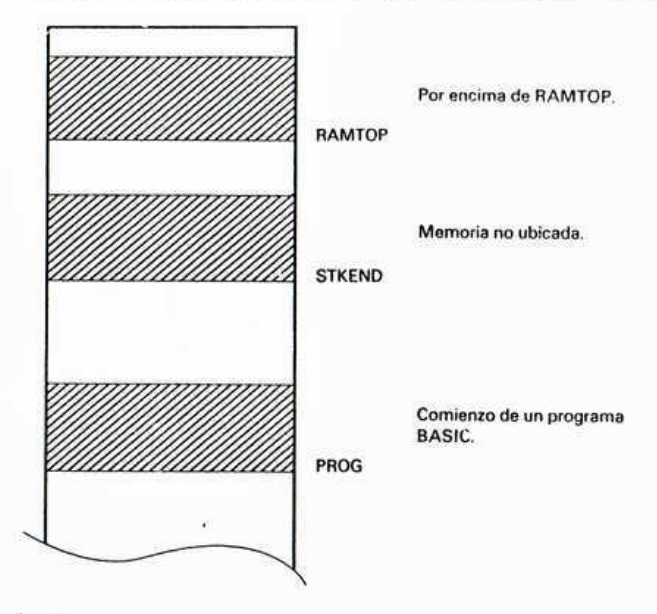


Figura 3.7

un programa que contiene dos o más grupos de sentencias que realizan la misma acción. Es una pérdida de tiempo y de memoria el mantener el mismo conjunto de sentencias repetidas en el mismo programa; sin embargo, como subrutinas sólo escribe una vez ese conjunto de sentencias como un bloque con propiedad propia. Este bloque separado de sentencias es lo que se llama una subrutina. Generalmente las subrutinas se escriben de forma independiente después del programa principal. Cuando se ejecuta el programa, cada vez que se necesite la subrutina se utiliza una instrucción especial de salto para comenzar la ejecución de dicha subrutina. Al final de ésta otra instrucción especial de salto hace que la computadora vuelva a la instrucción siguiente del programa principal que provocó el salto a esa subrutina.

El lenguaje ensamblador del Spectrum utiliza la instrucción CALL para provocar un salto a una subrutina. El CALL tiene que ir acompañado por la posición de memoria del primer byte de memoria de la subrutina. Como verá posteriormente, es factible dar un nombre o rótulo (etiqueta) a los bytes importantes de memoria; el rótulo de un byte de memoria puede utilizarse posteriormente en vez de la dirección numérica.

La figura 3.8 es un ejemplo de un pequeño programa que utiliza una subrutina para multiplicar una serie de números por 10.

```
10 REM go
20 REM org 23766
30 REM !uso de una subrutina
32 REM !cargar el primer número
35 REM ld a,(23760)
38 REM !salto a la subrutina
40 REM call Mul10
43 REM !guardar el resultado
45 REM ld (23763),a
48 REM !repetir con el segundo número
50 REM ld a,(23761)
55 REM call Mul10
60 REM ld (23764),a
62 REM !repetir con el tercer número
65 REM ld a,(23762)
70 REM call Mul10
75 REM ld (23765),a
80 REM ret;!fin del programa
   principal
85 REM !comienzo subrutina
90 REM Mul10;add a,a;!2 veces
95 REM ld b,a;!almacén temp
100 REM add a,a;!4 veces
105 REM add a,a;!8 veces
110 REM add a,b;!8 veces+2 veces
115 REM ret;!vuelta al programa
    principal
120 REM finish
```

Figura 3.8

La figura 3.9 es un pequeño programa BASIC que utiliza el programa de la figura 3.8. De momento, los detalles del programa no son relevantes; solamente debería observar que la ejecución de la subrutina se produce mediante la instrucción

CALL MUL10

donde MUL10 es un rótulo que indica dónde comienza la subrutina. Cuando se traduzca el programa a código máquina, el rótulo se convertirá en el número de la posición de memoria que contiene la primera instrucción de la rutina.

La última instrucción de toda subrutina es RET. La cual provoca que la computadora vuelva, o retorne, al lugar adecuado del programa principal. Todos los programas en código máquina del Spectrum se ejecutan con la función USR; como ésta es un salto a una subrutina, todos los programas en código máquina tienen que terminar con una instrucción RET.

```
1 REM 0000000000000000000000
  00000000000000000000000000000000
  00000000000000000000000000000
10 INPUT a,b,c
20 LET x=23760
30 POKE (x+0),a
40 POKE (x+1),b
50 POKE (x+2),c
60 RANDOMIZE USR 23766
70 FOR i=3 TO 5
80 PRINT PEEK (x+i)
90 NEXT i
```

Figura 3.9

Un programa puede utilizar cualquier número de subrutinas siempre que el comienzo de cada una de ellas esté claramente indicado y su última instrucción sea un RET. Un programa que contenga más de una subrutina es generalmente más fácil de comprender si todas ellas se colocan al final del programa principal, una después de otra.

Una de las ventajas de las subrutinas es que puede utilizar alguna que haya escrito otra persona sin necesidad de conocer los detalles de dicha subrutina. Una fuente muy útil de subrutinas para el Spectrum son las rutinas de la ROM del Sinclair. El Apéndice G proporciona una lista de aquellas rutinas y cómo deben utilizarse; la lista no está completa y sólo contiene las más sencillas de utilizar. pag 161.

4 ALGUNAS INSTRUCCIONES SENCILLAS

4.1 Los datos en la computadora

En este capítulo veremos las instrucciones en lenguaje ensamblador que permiten mover o trasladar los bytes de datos en el interior de la computadora y algunas operaciones aritméticas sencillas realizadas con números almacenados en un byte. Todos los datos que se utilizan en un programa en lenguaje ensamblador tienen que ser llevados por el programador a las posiciones de memoria adecuadas. La mayor parte de un programa en lenguaje ensamblador consiste en un conjunto de instrucciones las cuales permiten mover datos a los registros apropiados o a las posiciones de memoria.

Las instrucciones aritméticas sencillas suponen otro porcentaje muy elevado de un programa. Las instrucciones aritméticas más usuales son aquellas que permiten que el valor almacenado en una posición de memoria, bien un registro o un byte de memoria, pueda incrementarse o decrementarse en uno.

4.2 Carga de registros

Los datos que se están procesando o se procesarán en breve se almacenan en los registros de la unidad central de procesamiento. A los datos contenidos en un registro se puede acceder aproximadamente en la mitad de tiempo que el que supondría mover datos desde la memoria principal. Desde luego, en los registros solamente se puede almacenar una cantidad muy limitada de datos.

Un valor numérico se puede poner directamente en cualquiera de los registros sencillos de ocho bits, utilizando la instrucción con el siguiente formato:

```
LD r,n
```

donde n tiene el valor que se desea poner en el registro y r es uno de los registros de ocho bits A, B, C, D, E, H o L. Por ejemplo, la instrucción

```
LD D,20
```

pondrá el equivalente binario del número 20 como un número de ocho dígitos en el registro D. Si pudiera ver el contenido del registro D sería similar al de la figura 4.1.

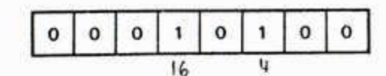


Figura 4.1

Normalmente el valor de un registro se tratará como un número entero positivo y tiene que estar comprendido en el rango que pueda contenerse en ocho bits, es decir, entre 0 y 255. La computadora también puede programarse para tratar el valor de un registro como si tuviera signo, o un número en complemento a dos. El rango de los números que pueden incorporarse en este caso está comprendido entre −128 y +127. Los números con signo se describieron en el Capítulo 2.

El registro A, llamado también acumulador, es el registro más importante para el programador. Todas las operaciones lógicas y la mayor parte de las operaciones aritméticas de ocho bits suponen la utilización del valor contenido en el acumulador y normalmente el resultado del procesamiento se deja asimismo en el acumulador.

Puesto que los registros se utilizan con frecuencia como posiciones de almacenamiento con propósito general para los datos que están esperando procesarse, surge la necesidad de mover datos entre registros. La instrucción que permite mover los datos entre registros es:

LD r1,r2

El efecto producido por esta instrucción es colocar una copia del valor contenido en el registro r2 sobre el registro r1. El valor del registro r2 no se ve afectado por esta instrucción. Los registros, r1 y r2, pueden ser cualquiera de los registros de ocho bits A, B, C, D, E, H o L. Por ejemplo, si el registro A contiene el valor 47 y el registro C el valor 127, la instrucción LD C,A colocará el valor 47 en el registro C y este valor permanecerá en el registro A. La figura 4.2 muestra el contenido de los registros antes y después de la instrucción.

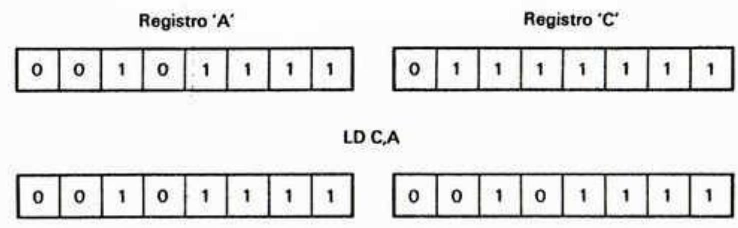


Figura 4.2

4.3 Incremento y decremento

Las operaciones aritméticas más usuales que requiere el programador son el incrementar o decrementar el valor de un registro o de una posición de memoria en uno. Para realizar estas operaciones hay dos instrucciones con el siguiente formato:

INC r y DEC r

Estas instrucciones producen el efecto de incrementar en uno o decrementar también en uno el valor de alguno de los registros de ocho bits A, B, C, D, E, H o L, o bien de alguna de las parejas registros de 16 bits BC, DE o HL. Estas instrucciones también pueden utilizarse para incrementar o decrementar el valor de una posición de memoria poniendo primero la dirección de la posición de memoria en la pareja de registros HL y utilizando más tarde dicha pareja de registros como un puntero. Por ejemplo, si la pareja de registros HL contiene el valor 32 000 y la posición de memoria 32 000 contiene el valor 58, entonces la instrucción INC(HL) incrementará el valor en la posición de memoria 32 000 en uno, es decir, a 59. Recuerde que los paréntesis es una forma abreviada de referirse al valor de una posición de memoria. La figura 4.3 muestra la utilización de la pareja de registros HL como un puntero.

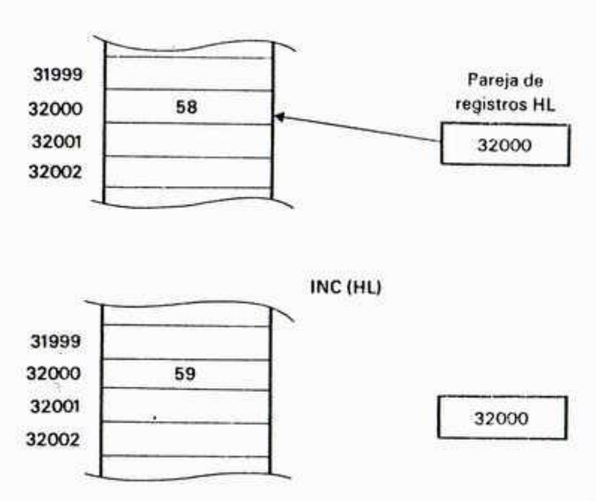


Figura 4.3

4.4 Transferencias de memoria

La mayoría de los datos utilizados por el programador estarán almacenados en memoria y será necesario transferirlos hacia y desde los registros del procesador central antes y después de su procesamiento. El acumulador, o registro A, es el registro que se utilizará con más frecuencia en el procesamiento, y es el que tiene mayor flexibilidad para mover los datos hacia y desde memoria.

Todas las instrucciones que utilizan datos en memoria contienen la dirección de la posición de memoria o utilizan una pareja de registros como puntero a la posición de memoria. La instrucción más sencilla para mover datos al registro A es:

LD A,(nn)

donde nn es la dirección de la posición de memoria que contiene el dato. Recuerde que aunque la posición de memoria puede contener solamente un byte, es decir, ocho bits de datos, la dirección de una posición de memoria requiere un número binario de 16 bits. Por tanto, nn es un número comprendido en el rango de 0 a 65 535.

De modo similar, los datos pueden moverse desde el registro A a memoria utilizando la instrucción:

LD (nn),A

No es muy corriente especificar directamente en una instrucción la dirección de la posición de memoria. Con frecuencia las direcciones se calcularán, en una parte previa del programa, y estarán contenidas en una pareja de registros de 16 bits. Las instrucciones de formato:

LD A,(rr)   y   LD (rr),A

donde rr es una de las parejas de registros BC, DE o HL, utilizan el valor contenido en el registro de 16 bits como la dirección de la posición de memoria que está involucrada en el movimiento. Como ya dijimos anteriormente para las instrucciones MOVE, el efecto de las mismas es copiar los datos desde memoria al acumulador, o viceversa, sin cambiar el valor de la fuente de los datos.

Anteriormente ya dijimos que la mayoría de los registros secundarios de la unidad central de procesamiento (CPU) tienden a utilizarse para fines específicos y el primero que veremos es la pareja de registros HL. Esta pareja forma un registro de 16 bits cuyo principal propósito es apuntar a las posiciones de memoria; en otras palabras, es el registro que normalmente se utiliza para contener las direcciones de los datos a mover hacia o desde memoria.

Mediante las instrucciones

LD r,(HL)   y   LD (HL),r

los datos se moverán entre cualquiera de los registros de ocho bits y memoria. De nuevo, mediante la pareja de registros HL utilizada como puntero de memoria, se puede cargar directamente una posición de memoria con un valor numérico. El formato de la instrucción es:

LD (HL),n

donde n tiene que ser un número comprendido entre 0 y 255.

4.5 Suma y resta

Ahora podemos ver algunas operaciones aritméticas sencillas. El microprocesador ZX80 Spectrum contiene las instrucciones que nos permitirán sumar o restar un valor del acumulador. Las instrucciones que realizan estas operaciones utilizando valores numéricos directamente son:

ADD A,n   y   SUB n

donde n es un número comprendido entre 0 y 255. Habrá observado que el formato de estas dos instrucciones es diferente. Esto es debido, a que si bien el registro A es el único de ocho bits que puede utilizarse con la instrucción ADD, las sumas de 16 bits pueden realizarse mediante la pareja de registros HL, pero la resta utilizando la instrucción SUB solamente puede realizarse utilizando valores de ocho bits y el registro acumulador. Esto quiere decir que en la instrucción ADD la utilización del registro A tiene que indicarse en la misma, pero esto no será necesario en la instrucción SUB puesto que el registro A es el que se utiliza siempre. Debe recalcarse que la instrucción SUB sería errónea si se escribiera de la misma forma que la instrucción ADD; no es opcional el omitir el registro A.

Cuando se realiza la suma o resta utilizando el registro acumulador, el resultado de dicho cálculo se deja en el acumulador. Por ejemplo, el efecto producido por las instrucciones

```
LD A,52
SUB 19
ADD A,22
INC A
```

es colocar el valor 52 en el registro acumulador, restar 19 del mismo dejando el valor 33 en el acumulador, sumar 22 a éste para que nos dé el valor 55 y, finalmente, incrementar éste en uno dejando el valor 56 en el acumulador.

El valor de cualquier registro de ocho bits, incluyendo al acumulador, puede sumarse a/o restarse del valor en el acumulador con las instrucciones:

ADD A,r   y   SUB r

Mediante la pareja de registros HL como puntero de memoria, tal y como mostrábamos anteriormente en este capítulo, se puede también sumar un valor en memoria a, o restar de, el valor del acumulador.

La figura 4.4 es un pequeño fragmento de un programa que muestra algunas de las instrucciones que hemos enumerado en este capítulo. La primera instrucción coloca el valor 32 500 en la pareja de registros HL. Después se copia el valor de la posición de memoria 32 500 al registro A utilizando HL como un puntero de memoria. La siguiente instrucción incrementa el valor de HL en uno, es decir, a 32 501 y luego se pasa el valor contenido en esta posición de memoria al registro B. Ahora se suman los dos valores y el resultado se copia desde el acumulador al registro B. A continuación el valor del acumulador se dobla, añadiéndose así mismo, y finalmente los valores de A y B se copian de nuevo a memoria.

```
 1Ø REM go
 2Ø REM org 2376Ø
 25 !cargar posición del
    primer elemento
 3Ø REM ld hl,325ØØ
 35 REM !cargar primer número
 4Ø REM ld a,(hl)
 5Ø REM inc hl
 55 REM !cargar segundo número
 6Ø REM ld b,(hl)
 65 REM !sumar ambos
 7Ø REM add a,b
 8Ø REM ld b,a
 85 REM !duplicar resultado
 9Ø REM add a,a
 95 REM !almacenar resultados
1ØØ REM ld (hl),b
11Ø REM dec hl
12Ø REM ld (hl),a
13Ø REM finish
```

Figura 4.4

Si al comienzo del programa la posición de memoria 32 500 contiene el valor 20 y la posición de memoria 32 501 contiene el valor 15, ¿cuáles serían los valores de estas dos posiciones de memoria al final del programa? Si tiene alguna duda sobre esta pregunta, la figura 4.5 muestra los valores de los registros y de las posiciones de memoria después de cada una de las instrucciones de este segmento de programa.

Instrucción	HL	A	B	Posición 32500	Posición 32501
LD HL,32500	32500	?	?	20	15
LD A,(HL)	32500	20	?	20	15
INC HL	32501	20	?	20	15
LD B,(HL)	32501	20	15	20	15
ADD A,B	32501	35	15	20	15
LD B,A	32501	35	35	20	15
ADD, A,A	32501	70	35	20	15
LD (HL),B	32501	70	35	20	35
DEC HL	32500	70	35	20	35
LD (HL),A	32500	70	35	70	35

Figura 4.5

4.6 Escritura de un programa

El programa que se muestra en la figura 4.6 es una subrutina muy práctica para el escritor de programas de juegos. Es un programa que producirá una línea de caracteres en la pantalla que se enrollará (scroll) hacia la izquierda. Es un programa adecuado para estudiar en este momento ya que la mayoría del programa consiste en movimientos de datos y en una aritmética muy sencilla.

La primera instrucción es un directivo (pseudoinstrucción), una instrucción al programa ensamblador para que reserve una posición de memoria. Ahora comienza el programa en cuestión cuando cargamos la pareja de registros HL con la posición en memoria del primer byte de datos para la línea que se enrolla (scroll). En el ejemplo mostrado es la línea superior. Utilizando los datos que se dan en un capítulo posterior en relación al fichero de visualización, será capaz de modificar este programa para enrollar cualquier línea. Para comprender el programa necesita saber que cada línea de caracteres de la pantalla se almacena en memoria como ocho bloques de 32 bytes y hay un bloque de 224 bytes entre cada uno de dichos bloques.

Si observa ahora el programa debería ser capaz de comprender

```
10 REM go
20 REM org 23760
22 REM Temp;defb 0
25 REM !programa para enrollar
   (scroll) la línea superior
30 REM ld hl,16384;!comienzo
   de la línea
40 REM ld c,8;!número de cols/líneas
50 REM Buclea
55 REM ld a,(hl)
60 REM ld (Temp),a;!guardar
   el primer carácter
70 REM ld b,31;!números de caracteres
   a mover
75 REM !bucle para mover
   caracteres
80 REM Bucleb
85 REM inc hl
90 REM ld a,(hl)
100 REM dec hl
110 REM ld (hl),a
120 REM inc hl
130 REM djnz Bucleb
140 REM inc hl
145 REM !poner el primero al final
    de la línea
150 REM ld a,(Temp)
160 REM ld (hl),a
165 REM !buscar siguiente parte
    de la línea
170 REM ld de,225
180 REM add hl,de
190 REM dec c
200 REM jp nz,Buclea
210 REM ret
220 REM finish
```

Figura 4.6

todas las instrucciones excepto la DJNZ Bucleb y la JP NZ, Buclea. JP NZ significa «saltar si la instrucción aritmética anterior proporcionó un resultado distinto de cero» y DJNZ es una combinación de las instrucciones DEC B y JP NZ.

4.7 Rótulos

Cuando está programando en BASIC frecuentemente se utilizan instrucciones que hacen referencia a otras instrucciones en el programa (por ejemplo, cuando se utilizan instrucciones GOTO). En BASIC, a cada línea del programa se le asigna un número de línea y dicho número es el utilizado para referirse a esta línea desde otra instrucción. Los programas en código máquina no tienen números de línea cuando se almacenan en memoria. Las instrucciones del programa que se refieren a otras instrucciones del mismo lo hacen mediante la dirección de la posición de memoria que contiene el primer byte de la instrucción.

El cálculo de la posición de memoria que contiene el primer byte de una instrucción determinada es una tarea dificultosa y lenta. Este problema se evita en el lenguaje ensamblador utilizando rótulos, que son nombres dados a instrucciones determinadas en el programa. Cuando se hace referencia a una instrucción mediante otra instrucción, se utiliza este rótulo. Cuando el programa se ensambla a código máquina, el programa ensamblador automáticamente convierte los rótulos en las direcciones de memoria adecuadas.

Al contrario que en BASIC, solamente se dan rótulos a aquellas instrucciones que estén referenciadas por otras. El rótulo lo elige el programador siguiendo las normas establecidas por el programador de ensamblador.

4.8 Programa

En este momento los programas que podemos escribir son muy limitados y tienen que ser simples subrutinas que pueden llamarse desde un programa en BASIC. Escribir una subrutina en lenguaje ensamblador que multiplique dos números mediante la suma del primer número por sí mismo el número de veces dado por el segundo. Por ejemplo, 4 × 3 es lo mismo que 4 + 4 + 4. Para escribir este programa necesitará la instrucción JP NZ, rótulo.

Ensamble el programa en la parte superior de la memoria y vuelva a reestablecer RAMTOP para dejar protegidas tres posiciones de memoria antes de comienzo de su programa. Por ejemplo, en una computadora Spectrum de 16K establezca RAMTOP a 32 499 y ensamble el programa para que comience en la posición 32 503 utilizando «org 32503». Los dos números a multiplicar estarán en las posiciones de memoria 32 500 y 32 501 y el resultado se debería colocar en la posición de memoria 32 502.

Escriba un programa BASIC que introduzca dos números y

realice una sentencia POKE de ellos a las posiciones 32 500 y 32 501. Después llame a la subrutina en código máquina y finalmente imprima el resultado extrayéndola mediante PEEK de la posición 32 502. Habrá observado en un programa anterior que la pareja de registros HL pueden cargarse directamente con un número (ver figura 4.4).

5 SALTANDO DE UN LADO PARA OTRO

5.1 ¿Por qué saltar?

Una de las cosas que sabrá de su programación en BASIC es que muy pocos programas comienzan en la primera instrucción y llegan hasta la última realizando cada instrucción solamente una vez y en el orden de sus números de línea. Todos, excepto los programas más sencillos, contendrán saltos que cambien el orden de ejecución de las instrucciones. Hay dos tipos de instrucciones de salto: el incondicional y el condicional.

5.2 Saltos incondicionales

En BASIC la instrucción de salto incondicional es GOTO número de línea; en lenguaje ensamblador la instrucción es

JP nn

donde nn es la dirección de una posición de memoria. El efecto de la instrucción es tomar la dirección nn como el primer byte de la siguiente instrucción a ejecutar. Generalmente en lenguaje ensamblador se reemplaza el número nn por un rótulo. Por ejemplo, en un programa que se ha ensamblado para que comience en la posición 32 500 de memoria y su primera instrucción es el rótulo «Princ», las instrucciones

JP 32500 y JP Princ

son equivalentes.

Los saltos incondicionales por sí solos no son muy prácticos porque generalmente provocan bucles infinitos, como se muestra en la figura 5.1. Desafortunadamente, en el Spectrum, si el pro-

```
10 REM go
20 REM org 23760
25 REM !un bucle infinito
30 REM ld a,1
40 REM Bucle;add a,a
50 REM rst 16
60 REM jp Bucle
70 REM finish
```

Figura 5.1

grama está en un bucle indefinido, la única forma de detener al programa es apagar la computadora y empezar de nuevo.

La instrucción JP permite a la computadora saltar a su siguiente instrucción en cualquier parte de la computadora; en la mayoría de los programas, casi todos los saltos se hacen a instrucciones que están unos bytes más allá de la instrucción de salto. Para aprovechar ésta, hay un segundo tipo de salto incondicional que permite realizar estos saltos pequeños. Es la instrucción de salto relativo, que solamente ocupa dos bytes de memoria comparados con los tres bytes utilizados por la instrucción JP. Se la llama de salto relativo porque la parte de datos de la instrucción especifica el número de posiciones de memoria desde el final de la instrucción de salto hasta el comienzo de la instrucción que seguirá al salto.

El formato de esta instrucción es:

JR n

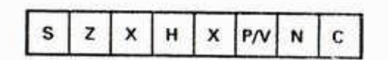
donde n es un número entre −128 y +128. De nuevo la instrucción a ejecutar después del salto se la puede referir mediante un rótulo, utilizando la instrucción JR rótulo. Cuando se ensambla el programa, el ensamblador calculará el número de bytes a saltar y reemplazará el rótulo por el salto relativo adecuado. Cuando utilice rótulos, tiene que comprobar que el salto requerido esté dentro del rango de la instrucción; de otra forma se producirá un error cuando se ensamble el programa. Si tiene alguna duda deberá utilizar la instrucción JP.

5.3 El registro indicador

Entre los registros del microprocesador Z80 hay uno de ocho bits que todavía no hemos tratado; es el registro F. Este no se utiliza para guardar datos sino un grupo de bits separados, muchos de los cuales se utilizan para guardar información relativa al resultado producido por las instrucciones aritméticas y lógicas. La figura 5.2 muestra los nombres dados a los diferentes bits.

Puesto que se utilizan para indicar el resultado de una instrucción previa, se les conoce como indicadores (flags). Dos bits interesantes del registro F son: el *indicador de signo* y *el indicador de cero.*

Si una operación aritmética o lógica produce un resultado negativo, el indicador de signo se pone a uno, mientras que un resultado de cero o positivo pone el indicador a cero. Si una instrucción produce un resultado de cero, el indicador de cero se pone a



Indicador		Prueba				
S	Signo	P	Positivo	o	M	Negativo
Z	Cero	Z	Cero	o	NZ	Distinto de cero
H	Acarreo-mitad	–				
P/V	Paridad/desbordamiento	PO	Paridad impar	o	PE	Paridad par
N	Suma/resta	–				
C	Acarreo	C	Acarreo	o	NC	Sin acarreo
X	Bits no utilizados					

Figura 5.2

uno; cualquier resultado distinto de cero, ya sea positivo o negativo, pone este indicador a cero. Posteriormente veremos el resto de los indicadores.

La figura 5.3 muestra un segmento de un programa con el valor de los indicadores de signo y de cero después de ejecutar cada instrucción.

5.4 Saltos condicionales

De su programación en BASIC sabrá que los saltos incondicionales tienen un uso limitado; casi siempre se utiliza con instrucciones que saltan dependiendo de los resultados de una prueba. Este tipo de saltos se conoce con el nombre de saltos condicionales.

Cuando un programa alcanza una instrucción de salto condicional o bien continúa con la siguiente instrucción al salto o salta a la instrucción del programa indicada. El salto sólo tiene lugar si se cumple la condición requerida; en el lenguaje ensamblador la condición va indicada por uno de los bits del registro indicador. La figura 5.4 muestra un pequeño programa que utiliza una ins-

	Indicadores	
Programa	Signo	Cero
LD A,10	?	?
SUB 11	1	0
LD B,A	1	0
INC B	0	1
INC B	0	0

Figura 5.3

```
10Ø REM go
```

```
 1Ø REM go
 2Ø REM org 2376Ø
 3Ø REM equ 3255Ø Comienzo
 4Ø REM ld hl,Comienzo
 5Ø REM ! Comienzo del bucle
 6Ø REM Bucle;ld a,(hl)
 7Ø REM ! Verificar si es cero
 75 REM sub Ø
 8Ø REM jr Z,Term
 9Ø REM add a,b
1ØØ REM ld b,a
1Ø5 REM inc hl
11Ø REM jr Bucle
12Ø REM Term;ld a,b
125 REM ld hl,Comienzo
127 REM !Guardar el resultado
13Ø REM ld (hl),a
135 REM ret
14Ø REM finish
```

Figura 5.4

trucción de salto condicional para salir de un pequeño bucle. El programa suma los números en posiciones de memoria sucesivas hasta que encuentra una posición que contiene un valor cero. Esta última posición se utiliza posteriormente para guardar la suma.

El formato de la instrucción de salto condicional es:

JP c,dir

donde c es la condición a verificar y dir es la posición de memoria de la instrucción a ejecutar si se cumple la condición. Por supuesto que la dirección normalmente se reemplazará por un rótulo en un programa ensamblador.

Algunas de las condiciones que pueden verificarse son:

Cero	JP Z,ROTUL1
Distinto de cero	JP NZ,ROTUL2
Negativo	JP M,ROTUL3
Positivo	JP P,ROTUL4

Estas condiciones las verifica la computadora comprobando el valor del bit adecuado del registro indicador.

En BASIC se puede saltar dependiendo de cualquier condición aritmética, pero en el lenguaje ensamblador se tienen que es-

BASIC	Ensamblador
10 LET A=10	LD A,10
20 PRINT A	BUCLE; RST 16
30 LET A=A+10	ADD A,10
40 IF A > 100 THEN 60	SUB 100
50 GO TO 20	JP P,FIN
60 STOP	JP BUCLE
	FIN; RET

Figura 5.5

cribir de nuevo como verificaciones del signo o de cero. De esta forma se puede expresar cualquier condición aritmética. La figura 5.5 muestra una pequeña sección de un programa BASIC y su equivalente en ensamblador. Mediante la resta y la comprobación del signo o de cero se puede programar cualquiera de las principales condiciones utilizadas en BASIC. La figura 5.6 muestra los equivalentes en lenguaje ensamblador de las cuatro condiciones principales. Mediante una combinación de comprobaciones, cualquier comprobación que se pueda programar en BASIC puede llevarse a cabo en lenguaje ensamblador.

También podemos realizar saltos condicionales relativos. Las condiciones que pueden verificarse por un salto relativo están limitadas, ya que pueden analizar el indicador de cero pero no el de signo. La instrucción con el indicador de cero es:

JR Z,desp y JR NZ,desp

donde desp es el número de bytes que separan a ésta de la instrucción que se ha de ejecutar si se cumple la condición. Esto, de nuevo, normalmente se reemplaza por un rótulo y el ensamblador calculará el número de bytes.

BASIC	Ensamblador
IF A=B THEN	SUB B JP Z,PROX
IF A > B THEN	SUB B JP P,PROX
IF A < B THEN	SUB B JP M,PROX
IF A < > B THEN	SUB B JP NZ,PROX

Figura 5.6

5.5 Comparaciones

Existe un problema al utilizar la sustracción para establecer los indicadores de condición antes de una instrucción de salto, ya que se modifica el valor original del acumulador mediante la sustracción. Con frecuencia es posible restaurar el valor mediante una adición, pero esto puede ser dificultoso e inconveniente. El microprocesador Z80 del Spectrum tiene una instrucción que evita este problema. Se llama instrucción de comparación y tiene el formato:

```
CP n
```

donde n es un valor de ocho bits, o el valor contenido en uno de los registros de ocho bits o el valor de una posición de memoria cuya dirección va en la pareja de registros HL. La instrucción admite comparar el valor contenido en el acumulador con otro valor de ocho bits sin modificar el valor del acumulador. La instrucción resta el valor especificado del valor contenido en el acumulador y coloca los bits del registro indicador de acuerdo al resultado. Posteriormente se ignora la resta y se deja el valor original en el acumulador.

El fin principal de la instrucción de comparación es determinar si el contenido del acumulador tiene un valor específico; con cierta frecuencia se utiliza después de una entrada por el teclado para comprobar si se ha introducido un carácter determinado. Después de introducir algo por el teclado el acumulador contendrá el código del carácter introducido. Por ejemplo, las instrucciones

```
CP 65
JR Z,AROTUL
CP 13
JP NZ,BUCLE
```

a continuación de una entrada del teclado comprobarán primero si el carácter introducido era una «A» y si así fue saltará al rótulo AROTUL. Si no era una «A» comprobarían si se ha pulsado la tecla «ENTER», y si no se hubiera pulsado, saltarían a la instrucción rotulada con BUCLE.

No todas las instrucciones modifican los bits del registro indicador; por ejemplo, la instrucción LD A,(HL) no modificará el registro indicador. Con frecuencia es práctico tener los bits del registro indicador de acuerdo al valor contenido en el acumulador incluso cuando la instrucción no afecta al registro indicador. La instrucción CP 0 puede utilizarse para modificar el registro indicador al comparar el acumulador con cero. El Apéndice A resume las instrucciones que afectan al registro indicador y qué bits se ven afectados. Ninguna de las instrucciones de movimiento de datos que hemos visto hasta ahora afectan al registro indicador; si queremos verificar el signo de un valor copiado desde memoria podríamos utilizar la instrucción CP 0, tal y como se muestra en el siguiente segmento de programa:

```
LD HL,32000
PROX;INC HL
LD A,(HL)
CP 0
JP M,PROX
```

Este programa continuará en el bucle hasta que encuentre un valor positivo en memoria.

5.6 Pseudooperaciones

Las pseudooperaciones son instrucciones en los programas en lenguaje ensamblador que no producen instrucciones equivalentes en código máquina. Los dos propósitos fundamentales de las pseudooperaciones son pasar información al programa ensamblador relativa a cómo se ha de traducir el programa y para comunicar al programa ensamblador que disponga espacio en memoria para los datos.

Algunas pseudooperaciones son válidas para un programa ensamblador, pero otras las entenderán la mayoría de los ensambladores. Los programas de este libro se han preparado utilizando el Ensamblador en Código Máquina del ZX Spectrum. Este ensamblador utiliza instrucciones escritas en sentencias REM de un programa BASIC y la primera instrucción de cualquier programa tiene que ser la pseudooperación

```
GO
```

y la última instrucción de todo programa tiene que ser la pseudooperación

```
FINISH
```

Estas instrucciones sólo las comprende este ensamblador; otra de las pseudooperaciones que tiene que utilizarse con este ensamblador y la mayoría de los ensambladores para el Spectrum es:

```
ORG nn
```

Esta instrucción comunica al programa ensamblador que la siguiente instrucción debería cargarse en la posición de memoria

nn. La mayoría de los ensambladores requieren esta instrucción al comienzo del programa para indicar al ensamblador dónde empezará en memoria. También puede utilizarse en medio de un programa para modificar la ubicación de la siguiente instrucción.

El Ensamblador en Código Máquina del ZX Spectrum admite otra forma muy práctica de la sentencia ORG; ésta es:

ORG a b

donde a es la posición de comienzo de memoria donde cargará el ensamblador al programa, pero se traduce el programa como si su dirección de comienzo estuviera en la posición b. En el Spectrum el mejor lugar para un programa en lenguaje máquina es la parte superior de la memoria, pero cuando se está ensamblando un programa en lenguaje ensamblador, la parte superior está ocupada por el programa ensamblador. Este formato de la instrucción permite que se traduzca el código en la parte inferior de la memoria y que se transfiera a la superior para ejecutarse.

La pseudooperación más útil es la instrucción DEFB. Esta instrucción indica al programa ensamblador que reserve las siguientes posiciones de memoria para almacenar valores de ocho bits y va seguida por los valores a colocar en estas posiciones. Cuando la instrucción va seguida de mas de un valor tiene que ir separada por espacios. Generalmente la instrucción va precedida por un rótulo y sí se puede referir a la primera posición por un nombre. La figura 5.7 es un pequeño programa que muestra la forma de utilizar esta instrucción para ubicar dos posiciones de memoria. El efecto producido por el programa es restar 32 del número almacenado en la posición MINUS y colocar el resultado en la dirección MAYUS. Se han elegido estos nombres porque este pequeño programa se puede utilizar para convertir las letras minúsculas a mayúsculas.

```
10 REM go
15 REM org 23760
20 REM Minus;defb 48
25 REM Mayus;defb 0
30 REM !
35 REM ld a,(Minus)
40 REM sub 32
45 REM ld (Mayus),a
50 REM ret
55 REM finish
```

Figura 5.7

5.7 Salida a pantalla

El Spectrum, como otros muchos microcomputadores, utiliza una técnica llamada mapa de memoria para producir la visualización sobre un televisor o la pantalla de un monitor. El mapa de memoria consiste en ubicar un área de la memoria de la computadora que contenga todos los detalles que se han de mostrar en la pantalla de visualización. Parte del programa monitor se utiliza para activar algunos circuitos de la computadora para copiar la imagen desde memoria a la salida de televisión y por tanto a la pantalla.

Una forma de producir una imagen en la pantalla es cargar los detalles de la imagen en memoria y después se visualizará automáticamente en la pantalla. El Spectrum realmente contiene dos áreas de memoria para la pantalla de visualización. Un área se utiliza para mantener las formas a visualizar y la otra para los atributos de color de la visualización. Debido a que el Spectrum es capaz de producir gráficos de alta resolución, la imagen que está contenida en memoria para las formas de cualquier carácter está contenida en ocho posiciones diferentes de memoria. Esto hace más complejo el colocar caracteres en la pantalla mediante la carga de sus atributos en memoria.

Sin embargo, puesto que la visualización de caracteres es un requisito usual, hay una subrutina en la ROM Sinclair que lleva a cabo esta función. Se puede llamar a esta función utilizando la instrucción:

RST 16

El efecto de esta instrucción es sencillamente utilizar el valor del acumulador como un código de carácter y visualizar este carácter en la pantalla. La rutina RST 16 es muy potente porque además de visualizar un carácter, dando su código de carácter, también reconocerá y actuará sobre los caracteres de control de la impresión.

Para controlar la entrada y salida, el sistema Spectrum utiliza una técnica conocida como canalización, que utiliza una parte independiente del programa ROM para cada una de las diferentes formas de salidas y entradas. El sistema Spectrum estándar utiliza cuatro canales:

Canal 1 — Admite entradas desde el teclado y la salida en la parte inferior de la pantalla.
Canal 2 — Admite la salida en la parte superior de la pantalla, pero no la entrada.

Canal 3 — Admite utilizar la zona de trabajo del BASIC; esto rara vez es práctico.
Canal 4 — Admite la salida a la impresora, pero no la entrada.

Cuando se enciende la computadora se inicializa el canal 1, esperando una entrada por el teclado. Para utilizar los otros canales hay que cargar el número del canal en el acumulador y llamar a una subrutina ROM para activar ese canal. La figura 5.9 muestra la forma de utilizar esta subrutina para obtener la salida en la parte superior de la pantalla. El Apéndice E muestra los códigos ASCII de los caracteres que pueden introducirse desde el teclado y después imprimirse, y el Apéndice F muestra los caracteres de control reconocidos por la rutina RST 16.

Las figuras 5.8 y 5.9 son programas que utilizan el RST 16 para imprimir en la mitad superior e inferior de la pantalla. La figura 5.8 es un programa muy sencillo, pero el de la figura 5.9 es un poco más complicado y muestra cómo se puede utilizar la rutina RST 16 para producir una imagen animada. En este momento esta instrucción nos dará la flexibilidad suficiente para la mayoría de las visualizaciones. La producción de imágenes en alta resolución lo discutiremos más adelante.

```
 1Ø REM go
 15 REM org 2376Ø
 17 REM !hazlo 1Ø veces
 2Ø REM ld b,1Ø
 25 REM Buclea;ld c,5
 27 REM !5 espacios al comienzo
    de la línea
 3Ø REM Bucleb;ld a,32
 35 REM rst 16
 4Ø REM dec c
 45 REM jr nz,Bucleb
 47 REM !cargar los códigos de
    las letras
 5Ø REM ld a,77;!  m
 55 REM rst 16
 6Ø REM ld a,1Ø1;! e
 65 REM rst 16
 7Ø REM ld a,11Ø;! n
 75 REM rst 16
 77 REM ld a,115;! s
 8Ø REM rst 16
 85 REM ld a,97;!  a
 9Ø REM rst 16
 95 REM ld a,1Ø3;! j
1ØØ REM rst 16
113 REM !pasar siguiente línea
115 REM ld a,13;! ENTER
12Ø REM rst 16
125 REM dec b
13Ø REM jr nz,Buclea
135 REM ret
14Ø REM finish
```

Figura 5.8

```
 1Ø REM go
 2Ø REM org 2376Ø
 3Ø REM ! Imagen animada
 4Ø REM ! Comienzo
 5Ø REM ld d,1Ø; ! número línea
 6Ø REM ld e,15; ! número col.
 9Ø REM ld b,127; ! carácter
    a mover
1ØØ REM call Print
11Ø REM ! Arriba y derecha
12Ø REM Ur;ld h,d
13Ø REM ld l,e; ! Guardar
    posición previa
14Ø REM Ur1;dec d; ! próxima línea a
    la derecha
142 REM ld a,d
145 REM ! parte superior pantalla
147 REM cp 1
15Ø REM jr z,Dr1
16Ø REM Ur2;inc e; ! próxima línea a
    la derecha
17Ø REM ld a,e
18Ø REM cp 31
185 REM ! lateral pantalla
19Ø REM jr z,U12
195 REM !imprimir sobre el
    antiguo e imprimir nuevo
2ØØ REM call Printer
21Ø REM jr Ur
22Ø REM ! Arriba e izquierda
23Ø REM Ul;ld h,d
```

```
240 REM ld l,e; ! guardar la
    posición previa
250 REM U11;dec d; próxima línea
    hacia arriba
252 REM ld a,d
255 REM ! parte superior pantalla
257 REM cp 1
260 REM jr z,D11
270 REM U12;dec e; ! próxima línea a
    la izquierda
272 REM ld a,e
275 REM ! lateral pantalla
277 REM cp 1
280 REM jr z, Ur2
290 REM call Printer
300 REM jr U1
310 REM ! abajo y derecha
320 REM Dr;ld h,d; ! Guardar
    posición previa
330 REM ld l,e
340 REM Dr1; inc d; ! próxima línea
    abajo
350 REM ld a,d
360 REM cp 21
365 REM ! parte inferior pantalla
370 REM jr z,Ur1
380 REM Dr2;inc e; ! próxima línea
    hacia la derecha
390 REM ld a,e
400 REM cp 31
405 REM ! lateral de pantalla
410 REM jr z,D12
420 REM call Printer
430 REM jr Dr
440 REM ! abajo e izquierda
450 REM D1;ld h,d
460 REM ld l,e; ! guardar
    posición previa
470 REM D11;inc d; ! próxima línea
    hacia abajo
480 REM ld a,d
490 REM cp 21
495 REM !parte inferior pantalla
500 REM jr z,U11
510 REM D12;dec e; ! próxima línea a
    izquierda
512 REM ld a,e
515 REM !lateral pantalla
517 REM cp. 1
520 REM jr. z,Dr2
530 REM call Printer
540 REM jr D1
550 REM Print;ld a,2
554 REM call 5633; !abrir canal
557 REM ld a,22
560 REM rst 16
570 REM ld a,d
580 REM rst 16
590 REM ld a,e
600 REM rst 16
610 REM ld a,b
620 REM rst 16
630 REM ret
640 REM Printer;ex de,hl
650 REM ld b,32
660 REM call Print
670 REM ex de,hl
680 REM ld b,127
690 REM call Print
700 REM ret
710 REM finish
```

Figura 5.9

5.8 Programa

Escriba un programa que visualice un cuadrado de 6 × 6 asteriscos «*» en el centro de la pantalla. La figura 5.10 muestra la salida requerida. Utilice caracteres de control para imprimir en la posición correcta de la pantalla. Utilice bucles para acortar el programa.

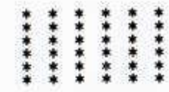


Figura 5.10

Como ejemplo un poco más complejo podría escribir un programa que produzca la salida que se muestra en la figura 5.11.

```
   *
  ***
 *****
*******
 *****
  ***
   *
```

Figura 5.11

6 UTILIZACION DEL TECLADO

6.1 Entrada desde el teclado

Muchos de nuestros programas requieren que los datos se introduzcan en la computadora desde el teclado. El programa que se utiliza para permitir que la computadora comprenda cuándo y qué tecla se ha pulsado del teclado es bastante largo y complejo, pero afortunadamente podemos utilizar las subrutinas de en ROM de Sinclair para realizar el trabajo pesado.

La principal subrutina comienza en la posición de memoria 703 y puede utilizarse mediante la instrucción:

CALL 703

El efecto producido por esta subrutina es analizar el teclado hasta que se pulse una tecla y cargar el código del carácter de esta tecla en la variable del sistema LAST-K, que se encuentra en la posición 23 560 de memoria. Esta contendrá el código del carácter de la última tecla pulsada hasta que bien se pulse una nueva o bien se cargue otro valor en esa posición.

La figura 6.1 es una pequeña rutina que introducirá un carácter desde el teclado. Cuando se pulse una tecla, cargará el código del carácter en el acumulador y luego sacará el carácter por la pantalla. Esta es la acción usual del BASIC. Si lo desea puede in-

```
  1Ø REM go
  2Ø REM org 2376Ø
1ØØØ REM Entsal;call 4264;!Rutin
     a ROM
1Ø1Ø REM cp 2Ø8;!comprobar tecla
     pulsada
1Ø2Ø REM jr z,Entsal;!no se ha
     pulsado ninguna
1Ø3Ø REM push af;!guardar carácter
     durante impresión
1Ø4Ø REM rst 16;!Eco de carácter
1Ø5Ø REM pop af;!recuperar carácter
1Ø6Ø REM ret;!fin de la rutina
1Ø7Ø REM finish
```

Figura 6.1

troducir un carácter desde el teclado, sin que se visualice en la pantalla; esto se realizaría eliminando la instrucción, RST 16, de la rutina.

El método anteriormente descrito es la forma más sencilla de llevar a cabo el proceso tan frecuente de introducir un carácter y visualizarlo en la pantalla. La ROM de Sinclair incluye otras rutinas que pueden utilizarse para introducir algo desde el teclado. Si está interesado en ellas, le dejaré que experimente con las rutinas de la ROM.

6.2 Códigos de carácter

Tengo que remarcar que la rutina dada en la sección anterior carga en el acumulador el código ASCII del carácter cuya tecla se pulsa. El Apéndice E muestra los códigos de caracteres usuales que pueden introducirse desde el teclado mediante esta rutina.

Hay que comprender, cuando se introducen números, que el código de carácter del número no es el mismo que el valor del número. Si se introducen números en la computadora, para utilizarlos en alguna operación aritmética, y luego se visualizan, tenemos que asegurarnos que utilizamos el código de carácter y el valor correspondiente en los lugares adecuados. Todos los procesos de entrada y salida utilizan el código de carácter y cualquier operación aritmética utiliza el valor del carácter. Cuando se utilicen códigos ASCII, como los utiliza el Spectrum, los códigos de carácter de los dígitos 0 a 9 pueden convertirse a su valor numérico restando 48 del código de carácter. Recuerde que tiene que sumar 48 de nuevo cuando vayan a sacar los resultados.

6.3 Entrada de números

Hasta ahora sólo hemos considerado cómo se pueden introducir y sacar números de un solo código. Esto es evidentemente muy restringido, y esta sección se encargará de las rutinas para introducir y sacar números con varios dígitos. La entrada de un número con más de un dígito no es tan sencilla como parece, incluso aunque la entrada esté restringida a números enteros.

Si el número a introducir es de tres dígitos, como 164, esto supondrá, desde luego, pulsar las tres teclas «1», «6» y «4». Esto podrá llevarse a cabo llamando a la subrutina de entrada y salida tres veces y cargar los códigos de carácter según se van introduciendo en posiciones de memoria sucesivas. Cuando se han introducido todos los dígitos del número, indicado normalmente por

otro carácter como la coma o ENTER, tienen que convertirse por separado los códigos de carácter a un valor en un solo registro o posición de memoria.

La primera etapa de esta conversión es cambiar el código de carácter de cada dígito por su valor numérico. El cálculo que hay que realizar ahora es multiplicar el primer dígito por 100, el segundo por 10 y después sumar estos productos con el tercer dígito. El cálculo para el número 164 es:

$$1 \times 100 + 6 \times 10 + 4$$

Un método más eficaz es llevar a cabo el cálculo de la siguiente forma:

$$(1 \times 10 + 6) \times 10 + 4$$

Después de convertir los códigos de carácter a sus valores de dígito, el primer dígito se multiplica por 10, se añade el siguiente dígito a este producto, el resultado se multiplica por 10 y se suma el tercer dígito a este resultado. La gran ventaja de este método es que se puede extender para abarcar la entrada de un número con cualquier cantidad de dígitos.

Ya hemos visto en un capítulo anterior algunas operaciones aritméticas sencillas, y recordará que sólo podíamos realizar sumas y restas. No existen instrucciones directas en el lenguaje ensamblador del Spectrum para la multiplicación y la división. Una forma sencilla de realizar una multiplicación es una suma repetitiva. Por ejemplo, 5×3 es lo mismo que $5 + 5 + 5$ y 6×10 es $6 + 6 + 6 + 6 + 6 + 6 + 6 + 6 + 6 + 6$. Este parece un método muy poco refinado, pero con un bucle es muy sencillo de programar y desde luego rápido.

La figura 6.2 presenta una subrutina que introducirá un número y lo convertirá a un valor almacenado en memoria. Aunque la rutina es general y admitirá un número con tantos dígitos como sean necesarios, el valor final se almacena en una sola posición de memoria, lo que quiere decir que el valor máximo que se puede introducir es 255. La rutina acepta cualquier entrada como un dígito hasta que se pulse la tecla ENTER; podría ampliarse fácilmente para verificar si la tecla pulsada es un dígito o ENTER.

```
1Ø REM go
2Ø REM org 2376Ø
3Ø REM !establecer la memoria
4Ø REM Número;defb Ø
5Ø REM !comienzo entrada de
   números
6Ø REM Numin;call Entsal
```

```
 7Ø REM !comprobar si ENTER
 8Ø REM cp 13
 9Ø REM ret z
1ØØ REM !cambiar código a valor
11Ø REM sub 48
12Ø REM ld b,a
13Ø REM !recup total previo
14Ø REM ld a,(Número)
15Ø REM !multiplicar por diez
16Ø REM Call Mul1Ø
17Ø REM add a,b
18Ø REM !guardar número nuevo
19Ø REM ld (Número),a
2ØØ REM jr Numin
21Ø REM !
22Ø REM !subrutina para multiplicar
    por diez
23Ø REM Mul1Ø;add a,a;! 2 veces
24Ø REM ld c,a
25Ø REM add a,a; ! 4 veces
26Ø REM add a,a;! 8 veces
27Ø REM add a,c;! 8 veces+2
    veces
28Ø REM ret
285 REM !
29Ø REM !subrutina de entrada
295 REM !
3ØØ REM Entsal;call 4274
32Ø REM cp 2Ø8
33Ø REM jr z,Entsal
335 REM push af
34Ø REM rst 16
38Ø REM pop af
39Ø REM ret
4ØØ REM finish
```

Figura 6.2

La salida del valor contenido en un registro es esencialmente la inversa del proceso anterior, pero es aun un poco más complejo que la entrada. La figura 6.3 es una subrutina para sacar el valor contenido en el acumulador.

```
 1Ø REM go
 15 REM org 2376Ø
```

```
 2Ø REM !subrutina para imprimir
    el número continuo en el registro A
 25 REM !guardar valor de A
 3Ø REM ld b,a
 35 REM !imprimir parte superior
    pantalla
 4Ø REM ld a,2
 5Ø REM call 5633
 55 REM !comienzo rutina principal
 6Ø REM ld a,b
 7Ø REM ld b,Ø
 8Ø REM Buclea;sub 1ØØ
 9Ø REM jp m,Centenas
1ØØ REM inc b;! contar número de
    centenas
11Ø REM jr Buclea
115 REM !imprimir número de
    centenas
12Ø REM Centenas;add a,1ØØ
13Ø REM ld c,a;!guardar resto
    del número
14Ø REM ld a,b
15Ø REM cp Ø;! número de centenas
16Ø REM jr z,Continuar
17Ø REM add a,48;!cambiar a
    código
18Ø REM rst 16
19Ø REM ld d,1;! poner indicador
    para mostrar impresión centenas
2ØØ REM Continuar;ld b,Ø
21Ø REM ld a,c
22Ø REM Bucleb;sub 1Ø
23Ø REM jp m,Decenas
24Ø REM inc b;!contar número de
    decenas
25Ø REM Bucleb
255 REM !imprimir número de decenas
26Ø REM Decenas;add a,1Ø
27Ø REM ld c,a;!guardar resto
    del número
28Ø REM ld a,b
29Ø REM cp Ø;!alguna decena
3ØØ REM jr nz,Próximo
31Ø REM ld a,d
```

```
32Ø REM cp 1;!¿se imprime un Ø?
33Ø REM jp nz,Dígitos
34Ø REM ld a,b
35Ø REM Próximo;add a,48;! código
36Ø REM rst 16
37Ø REM Dígitos;ld a,c
38Ø REM add a,48
39Ø REM rst 16
4ØØ REM ret
41Ø REM finish
```

Figura 6.3

6.4 Números negativos

Todos los números que hemos utilizado hasta ahora han sido enteros positivos y la rutina de entrada de números de la última sección sólo tiene en cuenta números positivos. Desde luego se puede programar la computadora para que reconozca números positivos y negativos. El método más sencillo de obtener un número negativo es utilizar la instrucción:

NEG

que tomará el complemento a dos del valor del acumulador. Si el valor contenido en el acumulador es un número comprendido entre 0 y 127, esta instrucción convertirá el valor en la representación del entero con signo de un número comprendido entre 0 y − 127, como se mostró en el Capítulo 2. Por ejemplo, si el acumulador contiene el equivalente binario del número 45 y se ejecuta la instrucción NEG, el acumulador contendrá la representación entera con signo del número −45. La figura 6.4 muestra el contenido del acumulador en forma binaria antes y después de ejecutar la instrucción NEG.

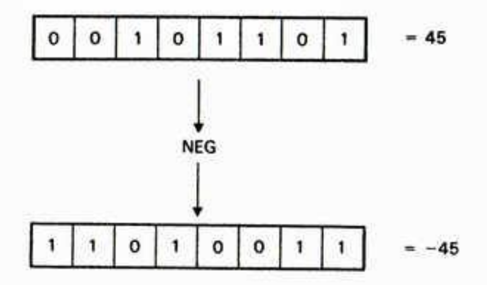


Figura 6.4

Cuando se utiliza la representación entera con signo el valor contenido en un registro de ocho bits o en una posición de memoria tiene que estar comprendido entre − 128 y + 127. Esta restricción en el rango de valores también se aplica al resultado de toda operación aritmética realizada con números con signo.

Ahora es necesario modificar nuestra rutina de entrada de números para que pueda introducir números negativos y positivos. La principal modificación se debe al primer carácter que será bien el carácter «−» o el «+» o un dígito. Si el primer carácter es un «+» o un dígito entonces el número se introducirá de la misma forma que en la sección anterior, pero si el primer carácter es un «−» después que se hayan introducido y convertido los dígitos a un valor positivo se ejecutará una instrucción NEG para convertirlo en un valor negativo. Dejaré la modificación de la rutina para que la realice como ejercicio.

6.5 Acarreo y desbordamiento

Cuando se realicen operaciones aritméticas utilizando datos de ocho bits todos los números tienen que estar comprendidos entre 0 y 255; si utilizamos enteros sin signo, o entre − 128 y + 127, si son enteros con signo. Ahora tenemos que ver cómo podemos verificar que nuestros resultados están comprendidos en los rangos admitidos.

Observando primero la suma de enteros sin signo, la figura 6.5 muestra dos ejemplos de sumas de números de ocho bits. En el primer ejemplo el resultado correcto puede expresarse como un número de ocho bits y por tanto la operación puede llevarla la computadora adecuadamente. El segundo ejemplo muestra de nuevo la suma de dos números de ocho bits, pero el resultado correcto necesita nueve bits y por tanto no puede obtenerse en un registro de ocho bits y la computadora produce un resultado erróneo. Por tanto, podemos comprobar si se ha producido el resulta-

```
               00101101       45
             + 10011001     + 153
               --------     -----
               11000110      198  Correcto

               10010001      145
             + 10011001     + 153
               --------     -----
Acarreo = 1    00101010       42  Erróneo
```

Figura 6.5

do correcto, ya que una suma que produzca un «1» en la novena posición coloca este bit en una de las posiciones del registro indicador o F. Este bit es el indicador de acarreo y se pone a uno si una suma o una resta produce un noveno bit; en otro caso es cero. La figura 6.6 proporciona ejemplos de un bit noveno, o de aca-

	10011001	153	
	− 10010001	− 145	
	00001000	8	Correcto
	10010001	145	
	− 10011001	− 153	
Acarreo = 1	11111000	248	Erróneo

Figura 6.6

rreo, producido por la sustracción. El valor de este bit puede comprobarse mediante una instrucción JP de las siguientes formas:

C indicador de acarreo a 1

o

NC indicador de acarreo a 0

El siguiente segmento de programa muestra una forma usual de utilización:

```
ADD A,B
JP C,ERROR
ERROR; !Comienzo de una rutina de error
```

También se puede utilizar el acarreo (C) y no acarreo (NC) con las instrucciones relativas (JR).

De igual forma que las instrucciones de suma y resta, el indicador de acarreo puede verse afectado también por las instrucciones de rotación y desplazamiento; éstas las veremos en el Capítulo 10.

Debido a que el indicador de acarreo es tan importante para la aritmética de la computadora, especialmente cuando se usan números que ocupan más de un byte, hay dos instrucciones que permiten modificar directamente el valor del indicador de acarreo. Estas son la SCF, que pone el indicador de acarreo a uno, y la CCF, que pone el indicador de acarreo al valor opuesto del que tuviera.

El registro indicador también contiene un bit conocido como indicador de desbordamiento, que se utiliza para indicar que el resultado de una operación aritmética queda fuera del rango permitido para los números con signo. Cuando se realiza una suma el desbordamiento solamente puede ocurrir si ambos valores son positivos o negativos, y la resta sólo puede producir desbordamiento si un número es positivo y el otro negativo. La figura 6.7 muestra varios ejemplos de sumas y restas que producen desbordamientos.

	01101010	+ 106	
	+ 01011110	+ 94	
	11001000	− 56	Respuesta errónea (hay desbordamiento)
	00011010	+ 26	
	+ 01011110	+ 94	
	01111000	+ 120	Respuesta correcta (no hay desbordamiento)
	10010110	− 106	
	+ 10100010	+ − 94	
Acarreo = 1	00111000	+ 56	Respuesta errónea (hay desbordamiento)
	01101010	+ 106	
	− 10100010	− − 94	
Acarreo = 1	11001000	− 56	Respuesta errónea (hay desbordamiento)

Figura 6.7

Si se produce desbordamiento se pone a uno el indicador de desbordamiento; en otro caso se pondrá a cero. El valor del indicador de desbordamiento se comprueba con una instrucción de salto condicional. Este indicador también tiene una segunda función, indicar la paridad de un byte y el nemotécnico para la condición indica este uso. No se preocupe de la paridad; solamente estamos interesados en la indicación de desbordamiento.

Las instrucciones que comprueba el indicador de desbordamiento son:

```
JP PE, rótulo
JP PO, rótulo
```

donde PE significa comprobación por no desbordamiento y PO comprobación por desbordamiento. No hay instrucciones de salto relativo (JR) que comprueben el indicador de desbordamiento.

La figura 6.8 es una subrutina que lee dos valores de memoria y los suma o los resta dependiendo del código de carácter de la

```
10 REM go
2Ø REM org 2376Ø
25 REM !establecer memoria
3Ø REM Núm1;defb Ø
35 REM Núm2;defb Ø
4Ø REM Sign;defb Ø
45 REM Result;defb Ø
5Ø REM !comienzo del programa
6Ø REM ld hl,Núm1
7Ø REM ld b,(hl);!primer número
8Ø REM inc hl;!apuntar segundo
   número
9Ø REM ld c,(hl);!segundo número
1ØØ REM ld a,(Sign)
11Ø REM cp 43;! un +
12Ø REM jr z,Sumar
13Ø REM cp 45;! un -
14Ø REM jr z,Restar
15Ø REM jr Error
155 REM !rutina para sumar
16Ø REM Sumar;ld a,b
17Ø REM add a,c
18Ø REM ld (Result),a
185 REM !Comp. desbordamiento
19Ø REM jp po,Error
2ØØ REM ret
21Ø REM !rutina para restar
22Ø REM Restar;ld a,b
23Ø REM sub c
24Ø REM ld (Result),a
245 REM !Comp. desbordamiento
25Ø REM jp po,Error
26Ø REM ret
265 REM !
27Ø REM !rutina de error
28Ø REM Error;ld a,2
29Ø REM call 5633
3ØØ REM ld a,69;! E
31Ø REM rst 16
32Ø REM ld a,114;! r
33Ø REM rst 16
332 REM ld a,114;! r
335 REM rst 16
34Ø REM ld a,111;! o
35Ø REM rst 16
36Ø REM ld a,114;! r
37Ø REM rst 16
38Ø REM ret
39Ø REM finish
```

Figura 6.8

tercera posición de memoria. El resultado se almacena en otra posición de memoria. Si se produce desbordamiento, se produce la visualización de la palabra E. La figura 6.9 muestra un programa BASIC sencillo que puede utilizarse para verificar esta subrutina.

```
   1 REM ØØØØØØØØØØØØØØØØØØØØØØØ
     ØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØ
     ØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØ
     ØØØ
  1Ø INPUT "¿Primer número?",a
  2Ø INPUT "¿Segundo número?",b
  3Ø INPUT "¿S(umar) o R(estar)?",X$
  4Ø POKE 2376Ø,a
  5Ø POKE 23761,b
  6Ø IF X$(1)="S" THEN POKE 237
     62,43: GO TO 9Ø
  7Ø IF X$(1)="R" THEN POKE 237
     62,45: GO TO 9Ø
  8Ø POKE 23762,Ø
  9Ø RANDOMIZE USR 23764
 1ØØ PRINT PEEK 23763
```

Figura 6.9

6.6 La instrucción EQU

La instrucción EQU es otra pseudooperación como la instrucción DEFB que vimos en el capítulo anterior. Aunque no produce ninguna instrucción en código máquina cuando se traduce, es muy útil para el prógramador de lenguaje ensamblador.

El formato para el Ensamblador a Código Máquina del ZX Spectrum es

EQU dirección nombre

y el efecto es permitir al programador dar un rótulo a posiciones de memoria incluso de fuera del programa en código máquina.

Esto hace que los programas sean más sencillos de leer y de comprender y es especialmente importante cuando se escriben programas bastante largos en diferentes secciones. El pequeño segmento del programa siguiente muestra una utilización típica:

```
EQU 5633 Canal
—
—
—
—
LD A,2
CALL Canal
LD A,13
RST 16
```

En este segmento se da el nombre de «Canal» a la posición de memoria cuya dirección es 5633 y posteriormente se la puede llamar por este nombre en vez de utilizar la dirección 5633. La utilización del nombre aclara el propósito de la instrucción mejor que utilizando la dirección.

6.7 Programa

Escriba un programa, utilizando las subrutinas desarrolladas en este capítulo, que pida dos números sacando primero una petición como:

INTRODUZCA EL PRIMER NUMERO
y después INTRODUZCA EL SEGUNDO NUMERO

El segundo número deberá ser positivo, pero el primero puede ser positivo o negativo. Multiplique ambos números utilizando la suma repetitiva. Si se produce desbordamiento se deberá sacar la palabra ERROR; en otro caso se debería sacar el resultado.

7 NUMEROS DE DIECISEIS BITS

7.1 Parejas de registros

La mayoría de los datos se utilizan y se transfieren en la computadora Spectrum como números de ocho bits. Sin embargo, a veces es más conveniente procesar los datos como números de dieciséis bits. En particular, cuando el proceso está relacionado con la determinación de la dirección de una posición de memoria, es esencial utilizar los 16 bits. Además de los registros específicos de 16 bits, que se utilizan para propósitos especiales y que explicaremos posteriormente, se pueden utilizar cualquiera de los registros de propósito general de ocho bits en parejas, para formar los registros de 16 bits BC, DE y HL. La pareja de registros HL se utiliza frecuentemente como un acumulador de 16 bits y rara vez se utilizan por separado como dos registros de ocho bits.

Todos los registros de 16 bits pueden cargarse directamente con un valor numérico mediante la instrucción:

```
LD dd,nn
```

donde nn es un número comprendido entre 0 y 65 535 y dd es uno de los registros de 16 bits BC, DE, HL, IX, IY o SP.

Las únicas instrucciones que permiten copiar datos de registro de 16 bits a otro son aquellos que están relacionados con el registro SP. Discutiremos este registro con detalle en el Capítulo 8. Los datos se pueden copiar de una pareja de registros a otra tratándolos como simples registros de ocho bits. Por ejemplo, para copiar el valor contenido en el registro HL sobre el registro BC utilizaríamos las siguientes instrucciones:

```
LD B,H
LD C,L
```

Solamente se pueden intercambiar los valores de los registros HL y DE mediante la instrucción:

```
EX DE,HL
```

7.2 Datos de dieciséis bits en memoria

Hay instrucciones que permiten mover datos entre memoria y las parejas de registros de 16 bits. Sin embargo, puesto que cada

posición de memoria sólo puede contener un valor de ocho bits, los valores de 16 bits tienen que acomodarse en dos posiciones sucesivas. Cuando se va a trasvasar un valor de 16 bits desde una pareja de registros a memoria, se mueven los ocho bits últimos a la primera posición de memoria y los ocho bits primeros a la segunda de las posiciones. La figura 7-1 muestra este proceso. El trasvase desde memoria a una pareja de registros se lleva a cabo

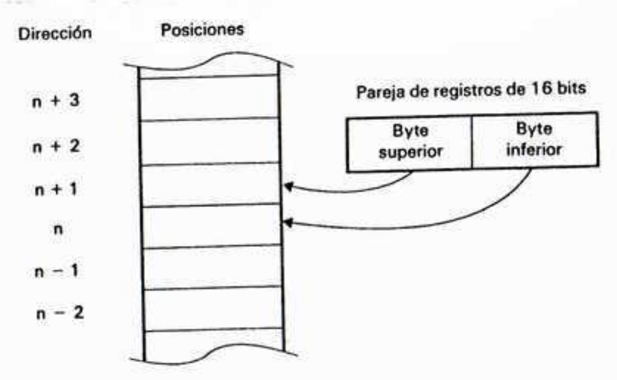


Figura 7.1

de la misma forma. Las instrucciones que realizan estos movimientos son:

LD dd,(nn) y LD (nn),dd

donde dd es una de las parejas de registros y nn es la dirección de la primera de las dos posiciones de memoria.

7.3 Más sobre el teclado

En el último capítulo vimos cómo utilizar la subrutina de la ROM del Spectrum para introducir un código de carácter en el acumulador. Todos los programas que quiera escribir en lenguaje ensamblador pueden utilizar esta subrutina para decodificar toda entrada desde el teclado. A veces quizá quiera utilizar el teclado de forma bastante diferente a la usual, es decir, un código diferente para cada tecla. Por ejemplo, en un programa de juegos puede decidir que cualquier tecla del lado izquierdo del teclado produzca un movimiento hacia la izquierda y cualquiera de la derecha lo haga hacia la derecha. Para obtener esta flexibilidad necesitamos examinar con más profundidad cómo obtiene el microprocesador los datos del teclado. Toda la información que entra en el procesador central, a excepción de los datos desde memoria, se introduce utilizando una de las instrucciones especiales para la entrada. La instrucción que se utiliza para las entradas desde el teclado es:

IN A,(n)

Cuando la computadora se encuentra esta instrucción, el valor de n, conocido como port o número de dispositivo, proporciona al sistema qué dispositivo externo o incluso qué parte del dispositivo se ha de analizar para la detección de datos. Estos, si hay alguno, se colocarán en el acumulador. Esta instrucción sólo introduce un byte de dato cada vez.

Analicemos ahora cómo ve el teclado del Spectrum al microprocesador y cómo podemos escribir programas para controlarlo. Cuando se pulsa una tecla se producen dos señales. Una se envía al port 254 y puede pasarse al microprocesador mediante la instrucción IN A,(254). Esta señal se produce tratando al teclado como ocho medias filas, cada una de ellas consistente en cinco teclas. Pulsando cualquiera de las teclas de una de estas filas pone uno de los bits de la señal de ocho bits a cero. Si no se pulsa ninguna tecla, todos los bits están a uno. La figura 7.2 muestra las

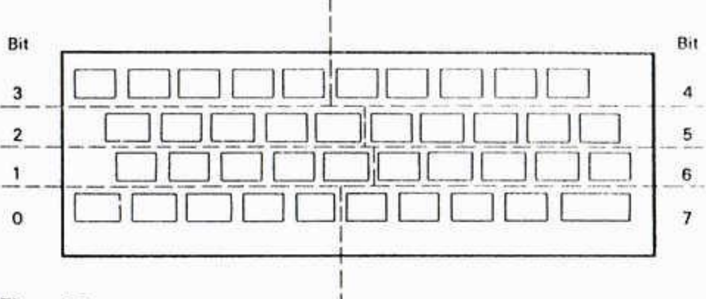


Figura 7.2

medias filas y el número del bit que se ve afectado por cada grupo de teclas. Puede observar que una tecla del lado izquierdo del teclado colocará uno de los bits 0, 1, 2 ó 3 a cero y una de la derecha colocará a cero uno de los bits del 4 al 7.

La figura 7.3 es un pequeño programa que imprime un carácter en el centro de la pantalla y lo mueve hacia la derecha o hacia la izquierda dependiendo de qué mitad del teclado se ha pulsado.

```
    1 REM ØØØØØØØØØØØØØØØØØØØØØØØ
   ØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØ
   ØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØ
   ØØØ
 1Ø REM go
 15 REM org 2376Ø
 18 REM !búsqueda parte izquierda
   derecha del teclado
 2Ø REM Buscar; in a,(254)
 25 REM cpl
 3Ø REM ld b,a;! guardar temp.
 35 REM and 15;!¿lado izquierdo?
 4Ø REM jr nz,Izquierdo
 45 REM ld a,b
 5Ø REM and 24Ø;!¿lado derecho?
 55 REM jr nz,Derecho
 6Ø REM jr Buscar
 65 REM !
 7Ø REM Izquierdo;ld a,2
 75 REM call 5633
 8Ø REM ld a,73;! I
 85 REM rst 16
 9Ø REM ld a,9Ø;! Z
 95 REM rst 16
1ØØ REM ld a,81;! Q
1Ø5 REM rst 16
11Ø REM ld a,46;! .
115 REM rst 16
117 REM ld a,13;! ENTER
118 REM rst 16
12Ø REM ret
125 REM !
13Ø REM Derecho;ld a,2
135 REM Call 5633
14Ø REM ld a,68;! D
145 REM rst 16
15Ø REM ld a,69;! E
155 REM rst 16
16Ø REM ld a,82! R
165 REM rst 16
17Ø REM ld a,46;! .
175 REM rst 16
19Ø REM ld a,13;! ENTER
195 REM rst 16
2ØØ REM ret
21Ø REM finish
```

Figura 7.3

7.4 Música con la computadora

Una de las posibilidades del Spectrum es producir una salida a través de su altavoz interno. El sonido se produce enviando pulsos eléctricos al altavoz; esto puede hacerse directamente en código máquina enviando datos al port de salida donde esté conectado el altavoz o de forma más sencilla utilizando la subrutina de sonido de la ROM del Spectrum.

La utilización del port de salida se discutirá en un capítulo posterior. La presente sección analizará la utilización de la subrutina de la ROM. Esta subrutina comienza en la dirección 949 de memoria y se la puede ejecutar mediante la instrucción:

CALL 949

Antes de poder utilizar esta subrutina se tienen que poner ciertos valores en las parejas de registros HL y DE. El valor del registro HL controla el intervalo de tiempo entre los pulsos enviados al altavoz, controlando de esta forma el tono de la nota que se va a producir. Cuanto más bajo sea el valor contenido en HL, mayor será la nota producida ya que el intervalo entre pulsos es más corto. El valor del registro DE controla la longitud de la nota —cuanto más alto es el valor contenido en DE, más larga será la nota producida.

La figura 7.4 muestra un programa muy sencillo que producirá un solo sonido. Practique con diferentes valores en DE y HL; los resultados pueden ser sorprendentes. También debería experimentar con un programa que modificase los valores de DE y HL mediante bucles; se pueden producir sonidos nada usuales y muy interesantes.

```
 1 REM ØØØØØØØØØØØ
1Ø REM go
2Ø REM org 2376Ø
3Ø REM ld hl,65Ø;! tono
4Ø REM ld de,1ØØ;! longitud
5Ø REM call 949;! rutina sonora
6Ø REM ret
7Ø REM finish
```

Figura 7.4

7.5 Modos de direccionamiento

Cuando escribíamos en BASIC escribíamos sentencias como

LET A = B + 5

sin preocuparnos de si nos referimos al valor contenido en la posición A o a la dirección de la posición A. Sin embargo, si escribimos un programa en lenguaje ensamblador tenemos que asegurar que especificamos los datos correctos en nuestras instrucciones. Los métodos utilizados para especificar los datos en una instrucción se conocen como modos de direccionamiento. En esta sección me gustaría recapitular los modos de direccionamiento que nos hemos encontrado hasta ahora y esclarecer las formas en que se especifican los datos.

El microprocesador Z80 tiene 10 modos de direccionamiento diferentes; ya hemos utilizado varios modos de éstos.

El modo de direccionamiento inmediato lleva el valor real del dato en la instrucción.

El modo de direccionamiento de registro utiliza el nombre de un registro como operando de la instrucción. En este registro se lleva el valor del dato.

El modo de direccionamiento extendido utiliza datos de memoria. El operando de la instrucción es la dirección de una posición de memoria que contiene el dato. En el lenguaje ensamblador el direccionamiento extendido se indica con paréntesis encerrando al operando. A este modo también se le llama direccionamiento indirecto.

El modo de direccionamiento relativo sólo se utiliza para las instrucciones de salto relativo, con o sin condiciones. El operando de estas instrucciones contiene el desplazamiento o el número de posiciones de memoria hasta la instrucción a ejecutar si se realiza el salto.

El modo de direccionamiento implícito no utiliza otra cosa que la propia instrucción para indicar el dato a utilizar. Las instrucciones que utilizan este modo de direccionamiento no tienen un operando independiente.

La figura 7.5 proporciona unos ejemplos de estos modos de direccionamiento y muestra también cómo se almacenan en memoria.

Instrucción	Código máquina	Direccionamiento
SUB 45	214, 45	Inmediato
INC B	4	De registro
LD A,(32000)	58, 00, 125	Extendido
JR C,BUCLE	56, 54	Relativo

Figura 7.5

Hay dos modos más de direccionamiento para discutir. El modo de direccionamiento indirecto con registro utiliza datos de memoria, como el modo extendido, pero el operando de la instrucción es una pareja de registros que contiene la dirección de la posición de memoria que contiene el dato. Por ejemplo, la instrucción

LD A,(HL)

toma el valor contenido en la pareja de registros HL como la dirección de una posición de memoria y después copia el valor contenido en esa posición en el registro A. Esto se muestra en la figura 7.6. Cuando se cargan datos en el registro acumulador se puede utilizar cualquiera de las parejas de registros de 16 bits

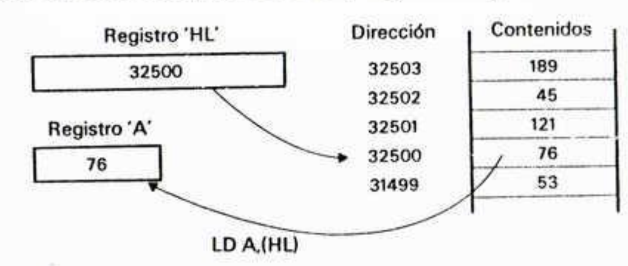
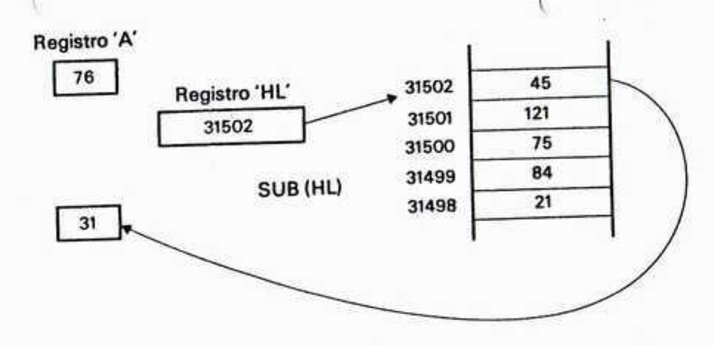


Figura 7.6

para contener la dirección de una posición de memoria, pero para cargar cualquier otro registro de ocho bits solamente se puede utilizar la pareja de registros HL para contener la dirección de memoria. Este modo de direccionamiento puede utilizarse también para instrucciones aritméticas o de salto como:

ADD A,(HL)
SUB (HL)
JP (HL)

Cuando se utilice para estas instrucciones solamente se puede utilizar la pareja de registros HL para contener la dirección de la posición de memoria. La figura 7.7 muestra algunos ejemplos de este modo de direccionamiento. De la misma forma que se pasan datos de memoria al acumulador o a cualquier otro registro de ocho bits, se puede utilizar este modo de direccionamiento para pasar datos desde registros a memoria. Se muestra un ejemplo de éstos en la figura mencionada. La flexibilidad extra obtenida de la utilización de la pareja de registros HL es la razón por la que se utiliza como el puntero principal de memoria.



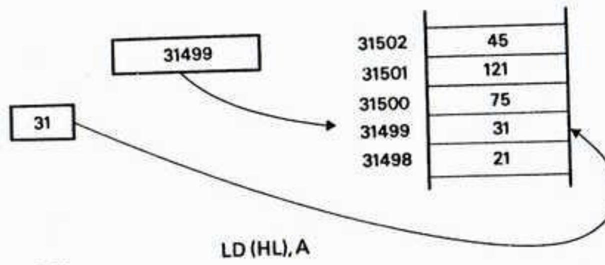


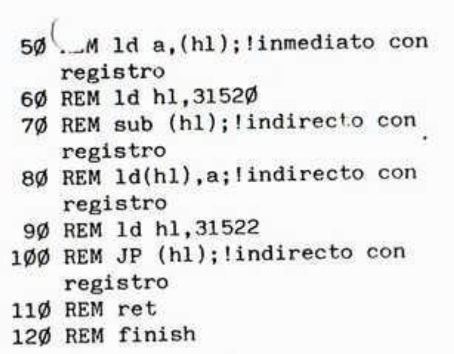
Figura 7.7

El modo de direccionamiento inmediato extendido es una extensión del modo de direccionamiento inmediato. Mientras el modo de direccionamiento inmediato utiliza valores de ocho bits, el modo inmediato extendido utiliza valores de 16 bits. Este modo se utiliza para poner directamente un valor en una de las parejas de registros de 16 bits. Por ejemplo, la instrucción

LD BC,15200

pondrá el valor 15 200 en la pareja de registros BC. Se puede cargar cualquiera de los registros de 16 bits BC, DE, HL, SP, IX e IY. La figura 7.8 es un pequeño segmento de programa que mues-

```
10 REM go
20 REM org 23760
30 REM !modo de dirección
40 REM ld hl,31500;!inmediato
   extendido
```

```
 50 ._M ld a,(hl);!inmediato con
    registro
 60 REM ld hl,31520
 70 REM sub (hl);!indirecto con
    registro
 80 REM ld(hl),a;!indirecto con
    registro
 90 REM ld hl,31522
100 REM JP (hl);!indirecto con
    registro
110 REM ret
120 REM finish
```

Figura 7.8

tra la forma de utilizar los modos de direccionamiento inmediato extendido y el indirecto con registro.

7.6 Programa

Escriba un programa que convierta las teclas numéricas en un teclado musical. Pulsando diferentes teclas sonarán notas diferentes. Tendrá que practicar para obtener un rango de notas que suenen bien. Al pulsar y mantener una tecla debería dar una nota sostenida; esto se puede conseguir haciendo que la duración de cada nota sea muy corta y haciendo un bucle de nuevo a la rutina de entradas desde el teclado. Tendrá que modificar la rutina de entradas para eliminar el eco.

8 REPETICIONES

8.1 Bucles

Todo programa que repita una sección del programa más de una vez está utilizando un bucle. Probablemente una de las características más importantes de cualquier computadora es su habilidad de hacer lo mismo muchas veces sin cometer errores; con otras palabras, su habilidad de llevar a cabo un bucle de programa. Podemos reconocer diferentes tipos de bucle mediante el método utilizado para determinar cuándo parar la ejecución del bucle.

El tipo de bucle más sencillo es aquel que no tiene fin, o bucle infinito. La figura 8.1 muestra un bucle sin fin sencillo que produce la visualización de una pantalla. No se debe utilizar normalmente un bucle sin fin puesto que sólo se puede detener apagando la computadora.

```
 1Ø REM go
 2Ø REM org 2376Ø
 3Ø REM equ 5633 Chanopen
 4Ø REM ld a,2
 5Ø REM call Chanopen
 6Ø REM ld a,31
 7Ø REM Bucle;inc a
 8Ø REM rst 16
 85 REM ld a,b
 9Ø REM jp Bucle
1ØØ REM ret
11Ø REM finish
```

Figura 8.1

Un método práctico de parar un bucle es realizar la comprobación de una condición específica como condición de terminar el bucle. El programa de la figura 8.2 introduce caracteres desde el teclado y los coloca en posiciones sucesivas de memoria hasta que se pulse la tecla ENTER. Las instrucciones del bucle se repiten hasta que la rutina de entradas introduce el código de carácter 13, el código de ENTER. La instrucción de comparación que va a continuación de la rutina de entradas producirá un resultado


```
 1Ø REM go
 2Ø REM org 2376Ø
 3Ø REM ld hl,32499
 4Ø REM !almacenar datos en 325ØØ
 5Ø REM Entra;inc hl
 6Ø REM call Entsal
 7Ø REM ld (hl),a
 8Ø REM cp 13;!es un ENTER
 9Ø REM jr nz,Entra
1ØØ REM Continúa;ret
11Ø REM !
12Ø REM Entsal;call 4264
14Ø REM cp 2Ø8
15Ø REM jr z,Entsal
16Ø REM push af
17Ø REM rst 16
18Ø REM pop af
19Ø REM ret
24Ø REM finish
```

Figura 8.2

de cero cuando se pulse ENTER, con lo que se ignorará el salto condicional, JP NZ, Entra, y el programa continuará con la instrucción rotulada «Continúa».

8.2 Bucles contadores

Una de las formas más sencillas de terminar un bucle es ejecutarlo un número determinado de veces. A éste se le llama un bucle contador y la figura 8.3 es un programa que muestra un

```
 1Ø REM go
 2Ø REM org 2376Ø
 3Ø REM ld b,5;!5 veces
 4Ø REM ld hl,325ØØ
 5Ø REM Bucle;call Entsal
 6Ø REM sub 48;!código a valor
 7Ø REM add a,(hl)
 8Ø REM ld (hl),a;!guardar el
    resultado
 9Ø REM djnz Bucle
1ØØ REM call Numsal
11Ø REM ret
12Ø REM !Entsal
```

```
13Ø REM !Entsal;call 4264
15Ø REM cp 2Ø8
16Ø REM jr z,!Entsal
17Ø REM push af
18Ø REM rst 16
23Ø REM pop af
24Ø REM ret
25Ø REM !
26Ø REM !Numsal;ld b,a
27Ø REM ld a,2
28Ø REM call 5633
29Ø REM ld a,b
3ØØ REM ld b,Ø
31Ø REM Buclea; sub 1ØØ
32Ø REM jp m,Centenas
33Ø REM inc b
34Ø REM jr Buclea
35Ø REM Centenas;add a,1ØØ
36Ø REM ld c,a
37Ø REM ld a,b
38Ø REM cp Ø
39Ø REM jr Z,Continúa
4ØØ REM add a,48
41Ø REM rst 16
42Ø REM ld d,1
43Ø REM Continúa;ld b,Ø
44Ø REM ld a,c
45Ø REM Bucleb;sub 1Ø
46Ø REM jp m.Decenas
47Ø REM inc b
48Ø REM jr Bucleb
49Ø REM Decenas;add a,1Ø
5ØØ REM ld c,a
51Ø REM ld a,b
52Ø REM cp Ø
53Ø REM jr nz,Próximo
54Ø REM ld a,b
55Ø REM cp 1
56Ø REM jr nz;Dígitos
57Ø REM ld a,b
58Ø REM Próximo;add a,48
59Ø REM rst 16
6ØØ REM Dígitos;ld a,c
61Ø REM add a,48
```



```
62Ø REM rst 16
63Ø REM ret
64Ø REM finish
```

Figura 8.3

ejemplo sencillo de un bucle contador. Este programa utiliza el registro B como un contador del número de veces que se ejecuta el bucle. La instrucción DJNZ decrementa el valor contenido en el registro B en uno y provoca un salto mientras este valor no sea cero. Cuando el valor de B llegue a cero ya se ha ejecutado el bucle el número de veces requerido y el programa continuará con la siguiente instrucción.

La instrucción DJNZ indica «decrementar y saltar si no es cero» y es equivalente a las dos instrucciones separadas siguientes:

DEC B
JR NZ, rótulo

La instrucción DJNZ sólo se puede utilizar con el registro B. Aunque se pueden utilizar otros registros, o incluso valores en memoria, como contadores de bucles, suele ser usual utilizar el registro B debido a la instrucción DJNZ.

El programa de la figura 8.3 se ha escrito con un valor en el registro B. Es mejor hacer las rutinas lo más generales posibles, para poderlas utilizar con otros programas. Este programa sería más práctico si el valor del contador del bucle se hubiera tomado de un byte de memoria. El valor en memoria lo podría haber colocado un programa principal que utilice la rutina.

Algunas veces el valor del contador del bucle se utiliza también como dato para su programa. La figura 8.4 muestra un pequeño programa que calcula la suma de números sucesivos.

```
1Ø REM go
3Ø REM call Entsal
4Ø REM sub 48
5Ø REM ld b,a
6Ø REM ld a,Ø
7Ø REM Bucle;add a,b
8Ø REM djnz Bucle
9Ø REM call Numsal
1ØØ REM ret
12Ø REM !
13Ø REM !Entsal;call 4264
15Ø REM cp 2Ø8
16Ø REM jr z,Entsal
17Ø REM ld b,a
```

```
180 REM RST 16
230 REM ld a,b
240 REM ret
250 REM !
260 REM Numsal;ld b,a
270 REM ld a,2
280 REM call 5633
290 REM ld a,b
300 REM ld b,0
310 REM Buclea;sub 100
320 REM jp m,Centenas
330 REM inc b
340 REM jr Buclea
350 REM Centenas;add,100
360 REM ld c,a
370 REM ld a,b
380 REM cp 0
390 REM jr z,Continúa
400 REM add a,48
410 REM rst 16
420 REM ld d,1
430 REM Continúa;ld b,0
440 REM ld a,c
450 REM Bucleb;sub 10
460 REM jp m,Decenas
470 REM inc b
480 REM jr Bucleb
490 REM Decenas;add a,10
500 REM ld c,a
510 REM ld a,b
520 REM cp 0
530 REM jr nz,Próximo
540 REM ld a,d
550 REM cp 1
560 REM jr nz,Dígitos
570 REM ld a,b
580 REM Próximo;add a,48
590 REM rst 16
600 REM Dígitos;ld a,c
610 REM add a,48
620 REM rst 16
630 REM ret
640 REM finish
```

Figura 8.4

Observe que en este programa se pone a cero el acumulador antes de utilizarlo para la suma. Esto es debido a que el valor, contenido en los registros o en posiciones de memoria, no estarán necesariamente a cero a no ser que los ponga a cero el programador.

Cuando se utiliza la instrucción DJNZ, se ejecuta el bucle hasta que el contador del bucle llegue al valor cero. Con frecuencia es más conveniente utilizar un contador de bucle hasta que llegue a otro valor que no sea cero. Cuando suceda esto, se utiliza una instrucción de comparación para detectar el final del bucle. La instrucción de comparación sólo se puede utilizar con el acumulador por lo que éste se utiliza como contador del bucle. La figura 8.5 es un pequeño programa que muestra esta forma de utilizar el acumulador.

```
 10 REM go
 20 REM org 23760
 30 REM !bucle con acumulador
        como contador
 40 REM ld hl,32500;apuntador
        de memoria
 50 REM ld a,10;valor de
        comienzo
 60 REM Bucle;ld b,(hl)
 70 REM inc hl
 80 REM ld (hl),b
 90 REM add a,b
100 REM cp 104
110 REM jr nz,Bucle
120 REM ret
130 REM finish
```

Figura 8.5

8.3 ¿Qué es una pila?

Una pila es un método particular de almacenar datos en posiciones de memoria sucesivas. El aspecto más importante de una pila es que sólo se pueden recuperar los datos de ella de forma inversa a como se almacenaron. Con otras palabras, el primer elemento que se puede recuperar es el último elemento almacenado. Otro término utilizado para la pila es una lista último-en-entrar-primero-en-salir.

Debido a la práctica que resulta una pila para almacenar datos, existe un registro especial en el microprocesador para contro-

...ar las pilas e instrucciones para almacenar y ...uperar datos de las pilas. Las pilas se utilizan de forma muy extensa en la ROM por el programa operativo de la computadora.

La figura 8.6 muestra cómo se almacenan las pilas en la memoria del Spectrum. El diagrama muestra que la pila se almacena

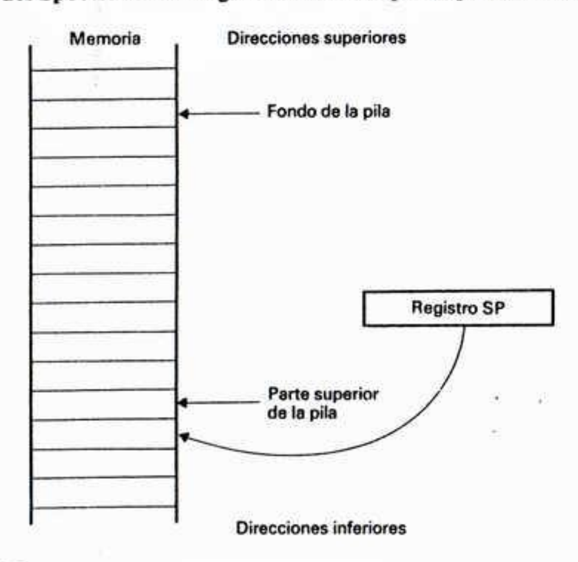


Figura 8.6

de arriba abajo, siendo la parte superior de la pila de dirección inferior al fondo. Se almacena de esta forma de tal manera que si la pila comienza en la parte superior de la memoria el tamaño de la pila puede continuar creciendo hasta que se llene la memoria.

Cada elemento de dato de la pila tiene una longitud de dos bytes. Cuando se añade o elimina un dato de la pila, el valor del apuntador de la pila, que siempre señala a la primera dirección disponible de la pila, se decrementa o incrementa en dos.

Las pilas se utilizan para almacenar datos temporales.

8.4 Utilización de las pilas

Una de las principales utilizaciones que hace el programa operativo de la computadora de la pila es controlar el retorno desde una subrutina al programa principal. Tendrá que analizar la figura 8.7 junto con . siguiente descripción. Cuando un progran.. alcanza una instrucción CALL se coloca en la pila el valor del contador del programa, que es la dirección de la siguiente instrucción, y después carga este contador con la dirección de la primera instrucción de la subrutina. Esto produce que se comience la subrutina. Al terminar la subrutina, la instrucción de retorno toma el valor superior de la pila y la coloca en el contador del programa. La figura muestra una subrutina en la dirección 16 800 que vuelve al programa principal en la instrucción cuya dirección es la 16 501. Antes de llamar a la subrutina el apuntador de la pila señala a la primera posición disponible. Cuando la subrutina alcanza la instrucción de retorno, tiene que estar la dirección de la instrucción adecuada en la parte superior de la pila; si se ha utilizado la pila en la subrutina entonces el programador tiene que

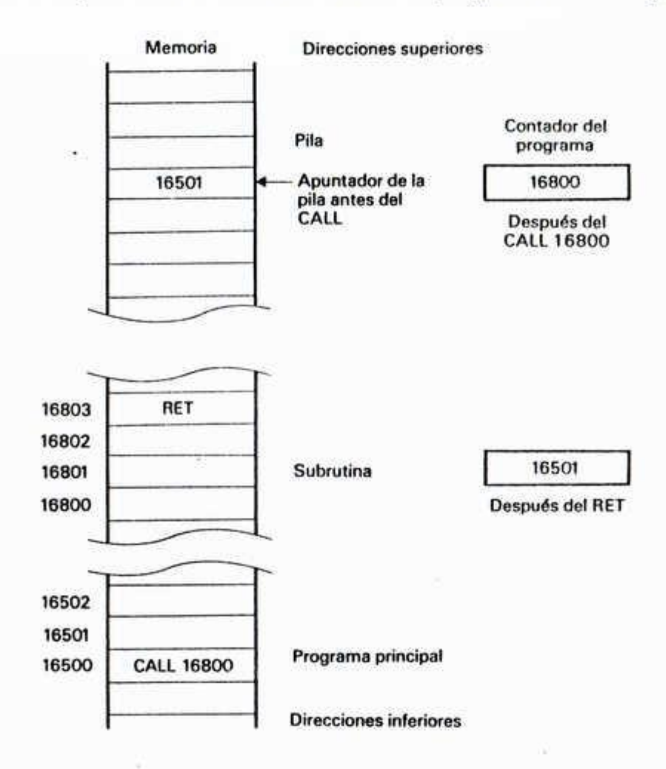


Figura 8.7

asegurar que se añaden y eliminan el mismo número de elemen-
os de datos desde la subrutina.

Sin embargo, es más importante ver cómo puede utilizar las pilas en sus propios programas. Una de las utilizaciones más usuales de la pila es guardar los valores contenidos en los registros del microprocesador durante la utilización de una subrutina. Al menos que se utilice un registro para transferir información entre la subrutina y el programa principal, todas las subrutinas deberían guardar en memoria los valores de todos los registros que se vayan a utilizar en la subrutina. Al final de la subrutina, justamente antes de volver al programa principal, pueden recuperarse los valores almacenados y reemplazados en los registros adecuados. Mediante esta técnica, cualquier programa puede utilizar la subrutina sin preocuparse de los cambios de contenido de los registros durante la ejecución de la subrutina. Cuando se utiliza una pila de esta forma es muy importante sacar los valores de la pila en orden inverso del que fueron almacenados.

Existen varias instrucciones que permiten al programador utilizar las pilas. En el Spectrum, los datos se añaden en una pila de dos bytes cada vez; a esta técnica se la conoce como empujar datos en la pila y la instrucción es:

PUSH rp

donde rp es cualquiera de las parejas de registros de 16 bits AF, BC, DE, HL, IX o IY. A la recuperación de datos de la pila se la llama sacar datos de la pila y la instrucción es:

POP rp

donde rp es cualquiera de las parejas de registros de 16 bits. Las instrucciones PUSH y POP no sólo almacenan o recuperan datos de un área de memoria ocupada por la pila, sino que también modifican los valores del registro SP.

La figura 8.8 muestra cómo utilizaría una pila una subrutina que utiliza los registros B, C y D para almacenar los valores en estos registros durante la ejecución de la subrutina.

En la mayoría de los casos, el programador utilizará la pila creada por el programa operativo de la computadora para almacenar datos. Pero como una pila es un método muy práctico de almacenar datos, el programador puede alguna vez crear una pila independiente para almacenar datos. Esto se puede hacer colocando la dirección de la parte superior de un área de memoria no utilizada en el registro SP. Por ejemplo, la instrucción

LD SP,20000

indicará una pila desde la posición 20 000 de memoria.


```
 1Ø REM go
    REM org 2376Ø
 3Ø REM Subrutina;push bc
 4Ø REM push de
 5Ø REM !parte principal de la
    subrutina
 6Ø REM !
 7Ø REM !
 8Ø REM !
 9Ø REM !fin del proceso
1ØØ REM pop de
11Ø REM pop bc
12Ø REM ret
13Ø REM finish
```

Figura 8.8

Debido a que no es posible pasar directamente datos de una pareja de registros de 16 bits a otra pareja de 16 bits, es más fácil muchas veces mover los datos utilizando una pila como memoria intermedia.

Hay también tres instrucciones que permiten almacenar datos en un registro de 16 bits para intercambiarlos directamente con el último elemento de la pila. El formato de la instrucción es:

EX (SP),rp

donde rp es uno de los registros HL, IX o IY de 16 bits.

8.5 Visualización de mensajes

La mayoría de los programas en alguna etapa de su proceso necesitan mensajes a la pantalla o a la impresora. En BASIC esto se hace mediante las sentencias PRINT y LPRINT con el mensaje encerrado entre comillas. Sin embargo, en lenguaje ensamblador la visualización de un mensaje es necesario hacerla en dos etapas. La primera consiste en almacenar el mensaje en el programa y después una sección independiente del programa envía el mensaje a la pantalla o a la impresora.

La primera etapa se cumplimenta mediante una de esas instrucciones especiales llamadas pseudooperaciones; esta vez la instrucción es:

DEFS

. instrucción va generalmente precedida po[illegible] rótulo y va seguida por el texto del mensaje. Una instrucción típica sería:

TITULO;DEFS Esto es un programa

El efecto de esta instrucción es almacenar el mensaje como una cadena de códigos de caracteres en posiciones sucesivas de memoria, comenzando en la posición rotulada con TITULO. Para sacar el mensaje a la pantalla se utiliza la subrutina de impresión, llamada por la instrucción RST 16, dentro de un bucle que cuenta el número de caracteres a visualizar. La figura 8.9 es un pequeño programa que visualiza el texto dado en el ejemplo de la instruc-

```
 1Ø REM go
 2Ø REM org 2376Ø
 3Ø REM Título;defs Esto es un
    programa
 4Ø REM push af
 45 REM push bc
 5Ø REM push hl
 55 REM ld a,2
 6Ø REM call 5633;!abrir canal
 65 REM !
 7Ø REM ld b,19;!número de letras
 75 REM ld hl,Título
 8Ø REM Bucle;ld a,(hl);!poner
    una letra en el registro A
 85 REM rst 16;!imprimirlo
 9Ø REM inc hl;!apuntar próxima
    letra
 95 REM djnz Bucle
1ØØ REM pop hl
1Ø5 REM pop bc
11Ø REM pop af
12Ø REM ret
13Ø REM finish
```

Figura 8.9

ción DEFS. Aunque esta rutina funciona perfectamente sería mejor si fuese más general para que pudiera utilizarse para imprimir cualquier texto. La figura 8.10 es una versión modificada de la rutina anterior. Toma la dirección de comienzo del texto en el registro HL, que se carga con esta dirección antes de llamar a la rutina; después saca los caracteres desde posiciones sucesivas de memoria hasta que se alcance una posición de memoria que con-


```
    REM go
 2Ø REM org 2376Ø
 3Ø REM Título;defs Esto es un
    programa
 4Ø REM defb Ø
 45 REM push af
 5Ø REM push hl
 55 REM ld a,2
 6Ø REM call 5633;!abrir canal
 65 REM !
 75 REM ld hl,Título
 8Ø REM Bucle;ld a,(hl);!poner
    una letra en el registro A
 85 REM cp Ø;!comprobar fin de
    mensaje
 9Ø REM jr z,Final
 95 REM rst 16;! imprimirlo
1ØØ REM inc hl
1Ø5 REM jr Bucle
11Ø REM Final;pop hl
115 REM pop af
12Ø REM ret
13Ø REM finish
```

Figura 8.10

tenga un valor cero. Se puede llamar a esta subrutina siempre que se quiera sacar un texto. Primero se tiene que almacenar en memoria el texto mediante la instrucción DEFS y ésta debe ir seguida inmediatamente de una instrucción DEFB para almacenar el valor cero en memoria.

8.6 Bucles anidados

Hemos visto la forma de escribir un conjunto de instrucciones que se ejecuten varias veces al colocarlas dentro de un bucle. Hay muchos programas en los que un bloque de instrucciones, que incluye un bucle, se repite varias veces. En otras palabras, tenemos un bucle dentro de otro bucle. A este tipo de estructura se la llama bucle anidado.

Hay varios tipos diferentes de bucles, cualquiera de ellos puede utilizarse bien como bucle interior o exterior. No existe ningún límite en el tipo o número de bucles que pueden utilizarse en una estructura de bucles anidados. La figura 8.11 es un sencillo programa que ilustra la programación de bucles anidados.

```
1Ø REM go
2Ø REM org 2376Ø
3Ø REM ld a,2
35 REM call 5633;!abrir canal
4Ø REM !
45 REM ld a,22;! AT
5Ø REM rst 16
55 REM ld a,11
6Ø REM rst 16
65 REM ld a,Ø
7Ø REM rst 16
75 REM !
8Ø REM ld c,6;! contador del bucle
   externo
85 REM Externo;ld a,23;! TAB
9Ø REM rst 16
95 REM ld a,9
1ØØ REM rst 16
1Ø5 REM !
11Ø REM ld b,8;! contador del bucle
   interno
115 REM ld a,143;! caracteres
   gráficos
12Ø REM l d,a;! memoria temp
13Ø REM Interno;rst 16
135 REM ld a,d
14Ø REM djnz Interno
145 REM !
15Ø REM dec c
155 REM jr nz,Externo
16Ø REM ret
17Ø REM finish
```

Figura 8.11

8.7 Programa

Escriba una subrutina que borre la pantalla mediante la visualización de 24 líneas, cada una de ellas formada por 32 caracteres.

Modificando la subrutina anterior, escriba un programa que visualice una pantalla llena de cada carácter imprimible y lo mantenga hasta que se pulse cualquier tecla.

9 LA PANTALLA

9.1 El fichero de la pantalla

Hasta ahora toda la salida a pantalla se ha llevado a cabo mediante la rutina de impresión de caracteres de la ROM del Spectrum. Hemos utilizado esta rutina sin considerar cómo produce la pantalla el Spectrum. Para sacar el mayor provecho de las posibilidades de visualización tenemos que analizar de forma detallada cómo se produce la visualización.

El Spectrum tiene dos áreas de memoria que utiliza para producir la visualización. La segunda de éstas está formada por el fichero de atributos la cual controla los valores de la visualización, ésta se describirá en el Capítulo 11. El área principal utilizada para visualizar es el fichero de la pantalla la cual controla las imágenes que se muestran en la pantalla. Para visualizar cualquier cosa sobre la pantalla es necesario cargar sobre el fichero de la pantalla una serie de números que especifican la imagen de la pantalla.

La pantalla de visualización del Spectrum tiene 24 líneas de 32 caracteres, cada uno formado por una matriz de 8 × 8 puntos; la figura 9.1 muestra algunos ejemplos de caracteres; producidos de esta forma. Cada fila de la matriz del carácter se almacena como un número de un byte en el fichero de la pantalla. La figura también muestra este número para cada fila de puntos de la matriz del carácter, en binario y en decimal. Al analizar los números binarios debería comprobar cómo se produce el número, uno indica la presencia de un punto y un cero indica que no lo hay. Cada carácter de la pantalla se almacena como un número de un byte, pero no se almacena en posiciones de memoria sucesivas. La pantalla se almacena en el fichero de la pantalla como tres secciones de ocho líneas; por tanto, la primera parte del fichero de la pantalla almacena los numeros que forman las matrices de caracteres de las primeras ocho líneas de la pantalla. Cada sección se almacena como muestra la figura 9.2. Los 32 bytes que forman las filas superiores de la matriz de carácter de los caracteres de la primera línea se almacenan en posiciones sucesivas de memoria seguidas de los 32 bytes producidos por las filas superiores de los caracteres de la segunda línea. Esto se repite hasta que se almacenan las filas superiores de las matrices de los caracteres de las ocho líneas. Después se repite este proceso para la segunda y pos-

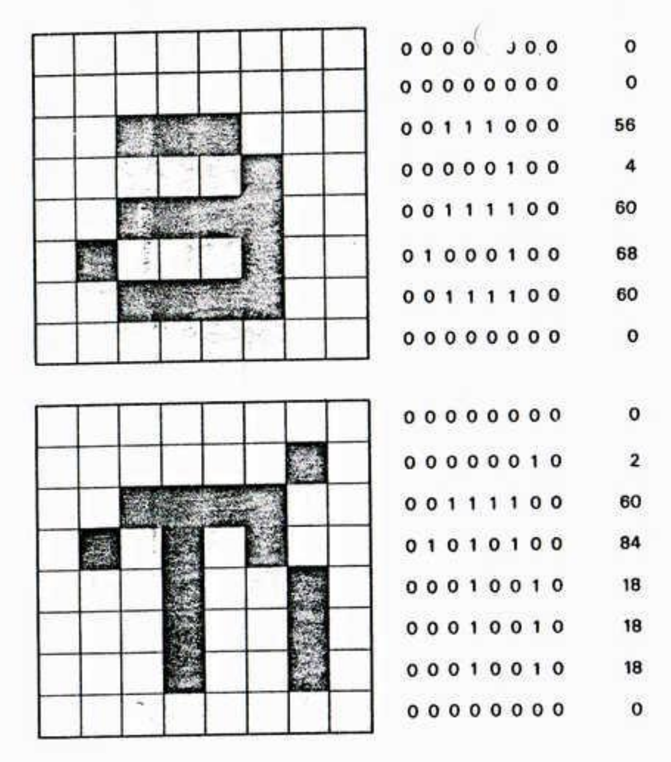


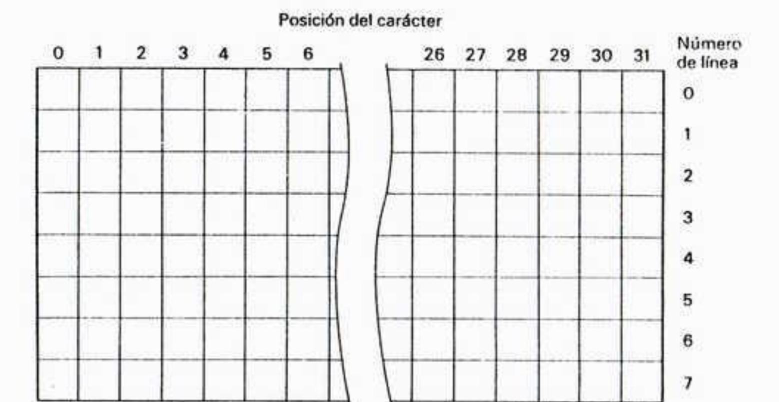
Figura 9.1

teriores filas de los caracteres de las ocho líneas, tal como muestra la figura 9.3.

9.2 Búsqueda de caracteres

Cada fila de la matriz de carácter se representa por un número binario de ocho bits, lo que significa que cada fila se almacena en una sola posición de memoria. Los números que constituyen las figuras de todos los caracteres estándar están almacenados en un bloque de memoria en la ROM del Spectrum, comenzando en la posición 15 616. Cada carácter se almacena como ocho números en posiciones sucesivas de memoria. Van almacenados por orden de código de carácter comenzando con el ESPACIO EN BLAN-





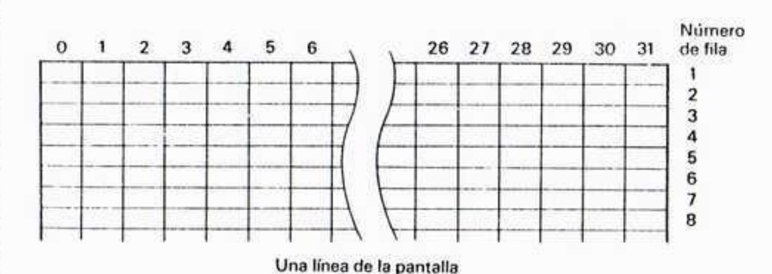


Figura 9.2

CO que es el código ASCII 32. Después va seguido por ocho bytes utilizados para conformar la figura del carácter cuyo código es 33, y así sucesivamente.

La posición del primer byte de cada carácter puede calcularse partiendo de su código de carácter mediante la fórmula:

(Código de carácter – 32) * 8 + 15616

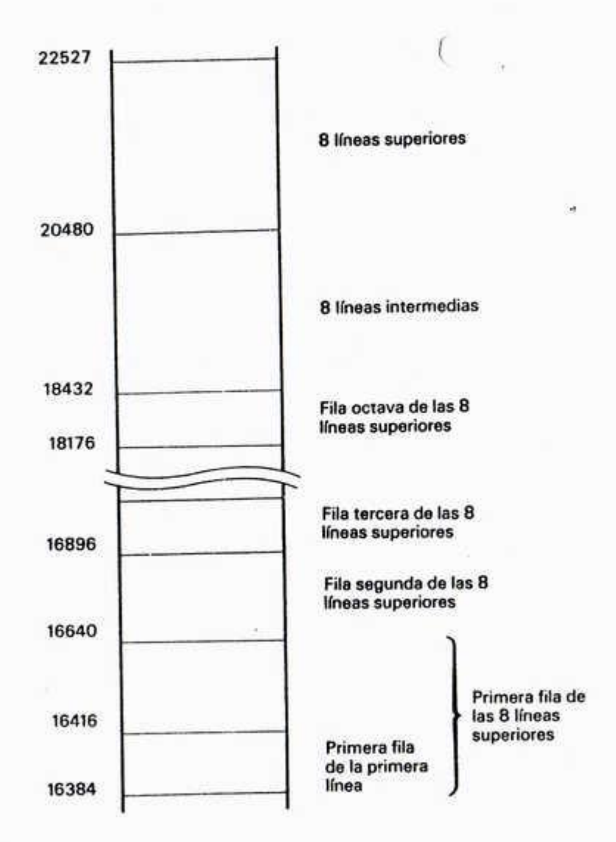


Figura 9.3

Los caracteres gráficos definidos por el usuario se almacenan de la misma forma excepto que se hace en la memoria RAM en vez de en la ROM; así puede modificarlos el programador. Normalmente se almacenan en la parte superior de la memoria, comenzando en la posición 32 600 en las máquinas de 16K y en la posición 65 368 en las de 48K. Una de las ventajas al utilizar el lenguaje ensamblador o el código máquina es que el programador puede establecer tantos caracteres definidos por el usuario como sean necesarios en cualquier área no utilizada de la memoria RAM.

9.3 Movimient[illegible] bloques

Todas las instrucciones del lenguaje ensamblador que hemos visto hasta ahora utilizan uno o dos elementos de datos, pero el microprocesador Z80 del Spectrum tiene instrucciones que permiten utilizar bloques completos de memoria.

El primer grupo de instrucciones permite mover el contenido de un bloque de memoria a otro bloque de memoria. Si recordamos que el fichero de la pantalla está almacenado en memoria, obviamente se pueden utilizar estas instrucciones para mover la pantalla o parte de la pantalla. Como un ejemplo, el segmento de programa de la figura 9.4 mueve la línea décima de la pantalla a la línea segunda. La instrucción importante es:

LDIR

que indica cargar, incrementar y repetir. Antes de ejecutar la instrucción, la pareja de registros HL tiene que contener la primera dirección del bloque que se quiere mover, la pareja de regis-

```
 1Ø REM go
 2Ø REM org 2376Ø
 3Ø REM equ 16414 línea2
 4Ø REM equ 18464 línea1Ø
 45 REM Temp:defw Ø
 5Ø REM !valores iniciales
 55 REM ld hl, línea1Ø
 6Ø REM ld de,línea2
 65 REM ld c,8;!cuenta filas/
    columnas
 7Ø REM Buclea;ld b,32:!bytes/
    línea
 75 REM Bucleb;ld a,(hl)
 8Ø REM ld (de),a;!mover a byte
 85 REM inc hl
 9Ø REM inc de
 95 REM djnz Bucleb
1ØØ REM push de;!memoria temp
1Ø5 REM ld de,224
11Ø REM add hl,de
115 REM ld (Temp),hl
12Ø REM pop hl
125 REM add hl,de
13Ø REM ex de,hl
135 REM ld hl,(Temp)
```

```
140 REM dec c
145 REM jr nz,Buclea
150 REM ret
160 REM finish
```

Figura 9.4

tros DE contendrá la dirección de la primera posición de memoria del bloque receptor y la pareja de registros BC contendrá el número de bytes de datos a mover. Puesto que el número de bytes de datos está contenido en una pareja de registros, el número de bytes que se pueden mover es 65 535. La acción de la instrucción LDIR es mover el contenido de una posición de memoria, señalada por la pareja de registros HL, a la posición de memoria señalada por la pareja de registros DE. Las parejas de registros HL y DE se incrementan después en uno y se decrementa en uno la pareja BC; este proceso se repite hasta que la pareja de registros BC sea cero.

Si hay un solapamiento entre los bloques de memoria puede haber problemas al utilizar la instrucción LDIR. Supongamos el siguiente segmento de programa que intenta mover de un bloque a un segmento bloque con dirección superior:

```
LD HL,30000
LD DE,30100
LD BC,500
LDIR
```

Cuando se ejecute este segmento se copiarán los primeros cien bytes del bloque original sobre el nuevo bloque, pero cuando el programa alcanza los segundos cien bytes a copiar ya se han sobrescrito. El efecto producido por este segmento será producir cinco copias de los primeros cien bytes del bloque original.

La instrucción que evita este problema es:

LDDR

que indica cargar, decrementar y repetir. LDDR funciona exactamente de la misma forma que la instrucción LDIR; excepto que en cada paso se decrementan en uno las parejas de registros HL y DE. Después de ejecutar la instrucción, las parejas de registros HL y DE apuntarán a las direcciones superiores de los bloques de memoria.

Hay dos instrucciones más que permiten copiar bloques de memoria. Estas son:

LDI y LDD

Son similares a instrucciones LDIR y LDDR en el sentido que ambas copian un byte de datos de un bloque a otro, los valores de los registros HL y DE se modifican para apuntar a las siguientes posiciones de los bloques respectivos, y la pareja de registros BC se decrementa en uno. Al contrario que las instrucciones previas, no se repiten automáticamente hasta que la pareja de registros BC contenga un cero. Después de copiar cada byte, se tiene que incluir en el programa la verificación del final de los bloques y se deberá hacer un salto de nuevo a las instrucciones LDI o LDD si no se han completado. Las instrucciones LDI y LDD se utilizan cuando no se conoce el número de elementos de datos a mover y el programa comprobará por el final del bloque.

Si está comprobando por un valor de cero en la pareja de registros BC, esto se muestra no por el indicador de cero, sino por el indicador de paridad/desbordamiento. Este indicador se pone a cero si BC es cero; en otro caso se pone a uno. Para comprobar si el indicador es cero la condición a verificar es PO y para uno la condición es PE.

9.4 Algunas rutinas de visualización

En esta sección me gustaría mostrarle algunas rutinas relativamente sencillas que permiten hacer scroll de la pantalla de diferentes formas. Encontrará que todavía no comprende todas las instrucciones de las rutinas; no se preocupe se explicarán próximamente.

La primera rutina hace scroll de la pantalla completa hacia la izquierda. El carácter de más a la izquierda de cada línea se mueve al final de la anterior y el primer carácter de la pantalla se pierde. Esto se muestra en la figura 9.5.

```
10 REM go
20 REM org 23760
30 REM equ 16384 Comienzo
40 REM equ 16385 Próximo
55 REM ! valores iniciales
60 REM ld de,Comienzo
65 REM ld hl,Próximo
70 REM ld b,192
75 REM !
80 REM Bucle;push bc
85 REM ld bc,31;!número de bytes
90 REM ld a, (de);!guardar primer
  byte
95 REM ldir;! mover una fila
```

```
100 REM dec hl
105 REM ld (hl),a;!wraparound
110 REM !hacer próxima fila
115 REM inc hl
120 REM inc hl
125 REM inc de
130 REM pop bc
135 REM djnz Bucle
140 REM ret
150 REM finish
```

Figura 9.5

La siguiente rutina hace scroll de la pantalla completa, sólo que esta vez lo hace hacia arriba. La pantalla se desplaza hacia abajo una línea a la vez, se muestra en la figura 9.6.

```
10 REM go
20 REM org 23760
30 REM Temp;defw 0
35 REM ld hl,22527;!sección
   inferior
40 REM call Secsc
45 REM call Linem
50 REM ld hl,20481;!sección
   intermedia
55 REM call Secsc
60 REM call Linem
65 REM ld hl,18431;!sección
   superior
70 REM call Secsc
75 REM call Blankl
80 REM ret;!fin del programa
   principal
85 REM !
90 REM Secsc;ld b,8;! líneas
95 REM Buclea;push bc
100 REM push hl
105 REM ld bc,224
110 REM ld de,32
115 REM scf
120 REM ccf
125 REM sbc hl,de
130 REM pop de
```



```
1(  REM lddr;!sección de
    movimiento
140 REM ld (Temp),hl
145 REM ex de,hl
150 REM scf
155 REM ccf
160 REM ld de,32
165 REM sbc hl,de
170 REM push hl
175 REM ld hl,(Temp)
180 REM scf
185 REM ccf
190 REM sbc hl,de
195 REM pop de
200 REM pop bc
205 REM djnz Buclea
210 REM ret
215 REM !
220 REM Linem;ld b,8
225 REM Buclelínea;push bc
230 REM ld hl,1824
235 REM add hl,de
240 REM ld bc,32
245 REM lddr;!mover a la siguiente
    sección
250 REM ld bc,32
255 REM ex de,hl
260 REM scf
265 REM ccf
270 REM sbc hl,bc
275 REM ex de,hl
280 REM scf
285 REM ccf
290 REM sbc hl,bc
295 REM pop bc;!recuperar contador
300 REM djnz Buclelínea
305 REM ret
310 REM !
315 REM Blankl;ld b,8
320 REM ld hl,16384
325 REM ld a,0
330 REM Bucled;push bc
335 REM ld b,32
340 REM Buclec;ld (hl),a
```

```
345 REM inc hl
35Ø REM djnz Buclec
355 REM ld de,224
36Ø REM add hl,de
365 REM pop bc
37Ø REM djnz Bucled
375 REM ret
38Ø REM finish
```

Figura 9.6

Finalmente, una rutina muy práctica de la ROM del Spectrum es la rutina de borrado de pantalla, que se realiza con el segmento de programa mostrado en la figura 9.7.

9.5 El margen de la pantalla

El color del margen de la pantalla puede modificarse de forma sencilla en cualquier momento mediante la instrucción:

OUT (254),A

donde el registro A contiene el valor del color del margen requerido.

La instrucción OUT se utiliza siempre que el procesador central de la computadora envía datos al mundo exterior. A todo dispositivo que está conectado a la computadora, bien para la entrada o salida de datos, se le asocia un port y cada port va numerado. Como puede observar, el port 254 controla el color del margen, pero éste también se utiliza para controlar el altavoz. También recordará que este mismo port se utiliza para la entrada del teclado.

Aunque la instrucción anterior cambiará el color del margen, el cambio será sólo temporal a no ser que el nuevo valor del color del margen se almacene también en los bits, tres a cinco de la variable del sistema BORDER. El resto de los bits de esta variable

```
1Ø REM go
2Ø REM org 2376Ø
3Ø REM ld b,24;!borrar 24 líneas
4Ø REM call 3652;!rutina ROM
5Ø REM ret
6Ø REM finish
```

Figura 9.7

del sistema con[illegible] la los atributos de la parte del fondo de la pantalla. El Apéndice G incluye los detalles de una subrutina para modificar el color del margen.

9.6 Programa

Se pueden obtener muchos efectos especiales mediante la manipulación directa del fichero de la pantalla. Escriba un programa que invierta las ocho líneas centrales de la pantalla. Será necesario utilizar otra área de memoria como un área de almacenamiento temporal para parte de la pantalla.

10 MULTIPLICACION Y DIVISION

10.1 Instrucciones de desplazamiento

Aunque ya hemos visto algunos métodos sencillos para multiplicar y dividir mediante las instrucciones ADD y SUB, este capítulo tratará sobre los métodos e instrucciones que permiten un procesamiento más eficiente. Las instrucciones que llevan a cabo la multiplicación y división se las llama desplazamientos.

Las instrucciones de desplazamiento mueven los bits de un registro o de una posición de memoria de un lugar hacia la derecha o hacia la izquierda. El microprocesador Z80 del Spectrum tiene tres instrucciones de desplazamiento: SRL, SRA y SLA.

La instrucción cuyo formato es

SRL m

indica «desplazamiento hacia la derecha» y m puede ser cualquiera de los registros de ocho bits o el contenido de la posición de memoria señalada por la pareja de registros HL. El desplazamiento lógico hacia la derecha trata el valor a procesar como un patrón de bits; mueve todos los bits un lugar hacia la derecha; se mueve un cero al bit final de la izquierda y el bit original del extremo derecho se pasa al indicador de acarreo. La figura 10.1

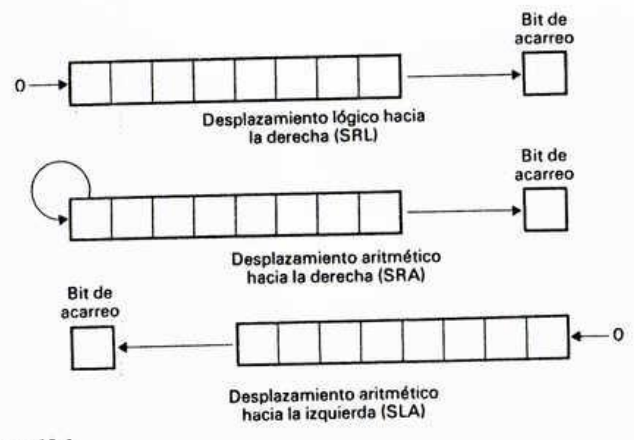


Figura 10.1

muestra la ope iva de todas las instrucciones de despla miento.

El bit que queda eliminado lo colocan todas las instrucciones de desplazamiento en el indicador de acarreo; después se pueden utilizar instrucciones de salto condicionales para comprobar el valor de este bit.

La segunda instrucción de desplazamiento es la SRA, cuyo formato es:

SRA m

que indica «desplazamiento aritmético hacia la derecha» y de nuevo m es cualquier registro de ocho bits o una posición de memoria. El efecto producido por el desplazamiento aritmético a la derecha es el mismo que el del desplazamiento lógico hacia la derecha excepto que el valor del bit izquierdo queda sin modificar. Un desplazamiento aritmético hacia la derecha es lo mismo que dividir el valor del registro o de la posición de memoria por dos, siendo el resto el indicador de acarreo.

La última instrucción de desplazamiento es la instrucción SLA, que tiene el formato:

SLA m

que indica «desplazamiento aritmético hacia la izquierda» y m tiene el mismo significado que en las instrucciones anteriores. La instrucción de desplazamiento aritmético hacia la izquierda mueve todos los bits hacia la izquierda y coloca un cero en el extremo de la derecha. La instrucción de desplazamiento hacia la izquierda produce el efecto de multiplicar el valor del registro o de la posición de memoria por dos.

Un desplazamiento lógico hacia la izquierda sería exactamente igual a un desplazamiento aritmético hacia la izquierda por lo que no hay necesidad de una instrucción diferente para el desplazamiento lógico hacia la izquierda.

10.2 Multiplicación

Al método utilizado para la multiplicación se le conoce por desplazar y sumar. Está basado en los mismos principios de la multiplicación y es más fácil de ilustrarlo trabajando sobre una multiplicación en binario; pero primero lo haremos con una multiplicación en decimal para recordar el método a nosotros mismos:

```
    1537    Multiplicando
  × 2054    Multiplicador
    6148    × 4
   7685     × 5
  0000      × 0
 3074       × 2
 3156998    Producto
```

La multiplicación en binario es igual que en decimal excepto que sólo necesita saber multiplicar por uno o por cero. Por tanto, lo verá mucho más fácil:

```
      10101
    × 11001
      10101    × 1
     00000     × 0
    00000      × 0
   10101       × 1
  10101        × 1
 1000001101    Producto
```

El método utilizado de desplazamiento y suma puede basarse de la siguiente forma:
Trabajando desde la derecha del multiplicador, sume el multiplicando al producto, si el bit del multiplicador es uno. Desplazar el multiplicando un bit a la izquierda y repetir el proceso para el siguiente bit del multiplicador hasta el final del multiplicador.

La figura 10.2 es un programa que multiplica el contenido de los registros B y C y deja el producto en el acumulador.

```
1Ø REM go
2Ø REM org 2376Ø
3Ø REM !Multiplicación por
   desplazamiento y suma
35 REM ld a,Ø
4Ø REM ld b,7
45 REM Bucle;slr c
5Ø REM !verificar si el bit es
   un 1
55 REM jp nc,Próximo
6Ø REM add a,b;add sumar al
   producto
65 REM Próximo;sla d,
```



```
7Ø REM djnz Bucle
75 REM ret
8Ø REM finish
```

Figura 10.2

Un método utilizado para la división está basado en la división desarrollada y es muy similar al método de «desplazar y sumar». Se le llama «desplazar y restar».

10.3 Rotaciones

Un grupo de instrucciones que son muy similares en la forma de operar a las instrucciones de desplazamiento son las instrucciones de rotación. La diferencia principal entre las instrucciones de desplazamiento y rotación es que las de rotación el bit que se elimina se coloca en el extremo contrario. De la misma forma que los desplazamientos, las rotaciones mueven los bits una posición hacia la derecha o hacia la izquierda.

Algunas de estas instrucciones hacen referencia al acumulador como parte de la instrucción y no como un operando separado. Estas instrucciones son instrucciones de un byte, mientras que el resto de las instrucciones de rotación, como las de desplazamiento, necesitan que se especifique el registro o la posición de memoria; estas son instrucciones de dos bytes.

Las rotaciones del acumulador pueden ser a la derecha o a la izquierda, y pueden o no incluir el indicador de acarreo en la rotación. Las cuatro instrucciones son:

RLA, RRA, RLCA y RRCA

La forma de operar de cada una de estas instrucciones se muestra en la figura 10.3

Las instrucciones de rotación para el resto de los registros de ocho bits o del contenido de una posición de memoria marcada por la pareja de registros HL son:

RL m	Rotación hacia la izquierda con mediación del indicador de acarreo.
RR m	Rotación hacia la derecha con mediación del indicador de acarreo.
RLC m	Rotación circular hacia la izquierda, sin mediación de acarreo.
RRC	Rotación circular hacia la derecha, sin mediación de acarreo.

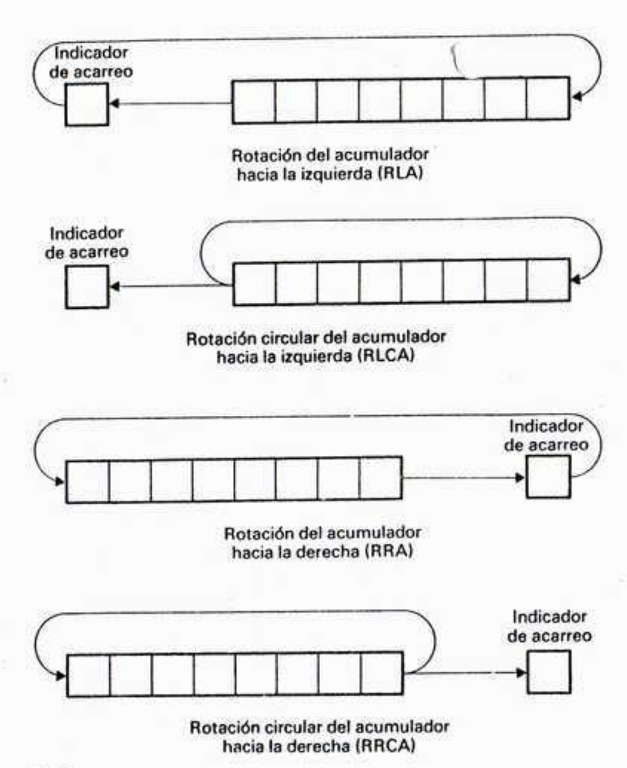


Figura 10.3

donde m es uno de los registros de ocho bits, excepto el acumulador, o una posición de memoria señalada por la pareja HL.

La forma de operar de estas instrucciones es exactamente igual a las mostradas para las correspondientes rotaciones del acumulador.

10.4 Programa

La multiplicación que se ha proporcionado anteriormente sólo trata números positivos; debería ser relativamente sencillo extenderla para que trate números con signo. Un método es primero multiplicar los valores positivos de los dos números y después utilizar las reglas de la multiplicación para determinar el

signo. La regla · que signos iguales proporcionan un resul[...] positivo y signos diferentes proporcionan un valor negativo. Recuerde que un número con signo con un uno en el bit del extremo izquierdo es negativo. Su valor positivo puede hallarse mediante la instrucción NEG.

Escriba una subrutina que realice la división por 10, después utilice esta subrutina para escribir otra subrutina que producirá la salida del valor contenido en el acumulador como un número comprendido entre −128 y +127. Los números negativos deben ir precedidos por un signo «–».

Utilizando la subrutina anterior y el programa de la multiplicación modificada, escriba un programa que pida números con signo de entrada y que calcule su producto. Una visualización típica sería:

```
4*-6 =-24
 7*8 = 56
```

Quizá quiera intentar un programa similar para la división, pero puesto que está trabajando con enteros probablemente le quedará un resto, por lo que debería visualizar el resultado y el resto.

11 LOGICA DE BITS

11.1 Operaciones con bits

Hasta ahora, todas las instrucciones que hemos visto han utilizado los datos en bytes. No hemos tratado instrucciones que utilicen menos de ocho bits de datos. Ahora vamos a ver algunas instrucciones que utilizan un bit de dato cada vez. Primero tenemos que numerar los bits de un registro, o de una posición de memoria, para poder nombrarlos. La figura 11.1 muestra que los bits van numerados de derecha a izquierda, comenzando por cero. No sé por qué se hace así, pero es estándar para todas las computadoras.

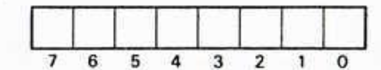


Figura 11.1

Todas las instrucciones de bit operan sobre cualquiera de los registros de ocho bits o sobre el contenido de una posición de memoria señalada por la pareja de registro HL. La primera instrucción tiene el formato:

BIT n,m

donde n es el número del bit y m es un registro de ocho bits o una posición de memoria. El propósito de esta instrucción es comprobar el valor del bit indicado y colocar el indicador de cero de acuerdo al resultado. Puesto que una instrucción de bit siempre va seguida por un salto condicional, un segmento típico de programa sería:

```
BIT 4,D
JPZ, PROXPART
```

si el registro D contiene el valor 45 (00101101 en binario), el bit 4, recuerde comenzando por la derecha desde cero, sería cero y el programa saltaría a la instrucción rotulada con PROXPART.

En el fichero de la pantalla todos los puntos que forman la imagen de la televisión están almacenados como bits y por tanto se puede utilizar la instrucción BIT para comprobar si un punto de la pantalla determinado está o no activo.

Las otras instrucciones que actúan sobre un bit en particular se utilizan para poner el valor del bit a uno o cero. La instrucción SET se utiliza para colocar el valor 1 en un bit en particular. La instrucción que pone un bit al valor 0 es la instrucción RES. El formato de estas dos instrucciones es:

SET n,m   y   RES n,m

donde n es el número del bit y n es el registro o posición de memoria. Ahora que ya somos capaces de trabajar con bits aislados podemos analizar una subrutina que produce un scroll suave de la pantalla al hacer scroll de un bit cada vez. La figura 11.2 es una rutina que hace scroll de la pantalla hacia la izquierda. Debería

```
 1Ø REM go
 2Ø REM org 2376Ø
 3Ø REM equ 16384 Fichpant
 4Ø REM !
 45 REM ld hl,Fichpant
 5Ø REM ld c,192
 55 REM ! Programa principal
 6Ø REM Filapróxima;ld a,(hl)
 65 REM sia a;!mover primer byte
 7Ø REM ld (hl),a
 75 REM !resto de la fila
 8Ø REM ld b,31
 85 REM Moverlínea;inc hl
 9Ø REM ld a,(hl)
 95 REM sla a;! scroll byte izquierdo
1ØØ REM ld (hl),a
1Ø5 REM !Comprobar si el primer bit era 1
11Ø REM jp nc,Nocarr
115 REM dec hl;byte previo
12Ø REM ld a,(hl)
125 REM set Ø,a;!poner bit a 1
13Ø REM ld (hl),a
135 REM inc hl
14Ø REM Nocarr;djnz Moverlínea
145 REM inc hl
15Ø REM dec c
155 REM jr nz,Filapróxima
16Ø REM ret
17Ø REM finish
```

Figura 11.2

...r capaz de modificar esta rutina para que se [illegible] a scroll hacia la derecha.

11.2 Instrucciones lógicas

El lenguaje ensamblador de Spectrum tiene un grupo de instrucciones que permiten cambiar los bits del acumulador con los bits de un registro de ocho bits o con una posición de memoria siguiendo las reglas de los operadores lógicos.

Los operadores lógicos operan sobre los correspondientes bits del acumulador y del operando. Los operadores lógicos son AND, OR, XOR y NOT. Las reglas para la combinación de bits se muestran en la figura 11.3. Excepto el operador NOT, el resto combinan las parejas correspondientes de bits y producen un resultado de un bit por cada pareja.

Las instrucciones lógicas son

AND   OR   XOR

y su formato es

AND p

Operador AND

0 AND0 = 0
0 AND1 = 0
1 AND0 = 0
1 AND1 = 1

Operador OR

0 AND0 = 0
0 AND1 = 1
1 AND0 = 1
1 AND1 = 1

Operador XOR

0 XOR0 = 0
0 XOR1 = 1
1 XOR0 = 1
1 XOR1 = 0

Operador NOT

NOT0 = 1
NOT1 = 0

Figura 11.3

donde p es un valor de ocho bits como un número, un registro de ocho bits o el valor contenido en una posición de memoria señalada por la pareja de registros las HL. La figura 11.4 proporciona algunos ejemplos de cómo funcionan instrucciones lógicas. Cuando una de éstas se ejecuta los indicadores de signo y de cero se modifican de acuerdo con el valor que quede en el acumulador.

Contenido de A	01011100
Contenido de B	11001100
Contenido de A después de AND B	00001100
Indicador de signo = 0	Indicador de cero = 0
Contenido de A	01011100
Contenido de B	11001100
Contenido de A después de OR B	11011100
Indicador de signo = 1	Indicador de cero = 0
Contenido de A	01011100
Contenido de B	11001100
Contenido de A después de XOR B	10010000
Indicador de signo = 1	Indicador de cero = 0
Contenido de A	01011100
Contenido de A después de CPL	10100011

Figura 11.4

La operación NOT la realiza la instrucción CPL. La instrucción CPL trabaja solamente sobre el acumulador y su efecto es modificar el valor de todos los bits del acumulador; todos los unos se convierten en ceros y todos los ceros a unos.

11.3 Datos empaquetados

Usted, como programador, debería siempre tratar de minimizar la cantidad de memoria utilizada por sus programas y sus datos. Cuanto menos memoria utilice, tendrá más espacio bien para un programa más comprensivo o para un número mayor de datos. Recientemente me mostraron un programa de una lista de correos para el ZX81 de 16K. Era un programa que tenía todo tipo de facilidades que se podrían requerir de un programa de este tipo, pero tenía un grave inconveniente: era demasiado grande. Cuando el programa estaba en memoria solamente dejaba espacio libre como para 10 ó 12 nombres y direcciones.

Mediante el lenguaje ensamblador se puede reducir el tamaño de los programas, pero ¿cómo se puede reducir el espacio utilizado por los datos? Con frecuencia la cantidad de datos que necesitamos es superior que los que puede contener una sola posición de memoria. En una sola posición de memoria podemos guardar

número comprendido entre 0 y 225 o entre -128 y +127. Supongamos que uno de los elementos de datos es la edad de una persona; evidentemente no necesitamos números negativos, pero incluso el rango de 0 a 225 es más amplio de lo necesario. Si sólo utilizamos siete de los ocho bits para la edad, podríamos escribir un rango de 0 a 127, que realmente es suficiente para cualquier aplicación, y podemos utilizar el bit octavo para que contenga otro dato. En efecto, en un bit podemos guardar cualquier elemento de dato que sólo tenga dos valores posibles, como, por ejemplo, si la persona es hembra o varón, si son o no son socios, si son clientes o suministradores. Cualquiera de estos elementos de datos podrían almacenarse como el bit octavo de la posición de memoria que contiene la edad. A esto se le llama empaquetación de datos.

La figura 11.5 muestra un ejemplo típico de una lista de socios de un club, donde debido a la edad de sus asociados, de 18 a 65

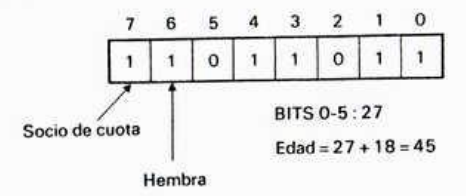


Figura 11.5

años, se utilizan los bits del 0 al 5 para que contengan la edad como el número de años que sobrepasen a 18, el bit 6 se utiliza para contener el sexo del socio y el bit 7 se utiliza para saber si la suscripción del socio requiere una cuota o no. El ejemplo muestra el caso de una mujer con cuota de 45 años. En un club con 1.000 socios, mediante este método, utilizaríamos 1.000 bytes de memoria, en vez de los 3.000 que se utilizarían si en cada elemento de dato se hubiese almacenado una posición independiente de memoria.

11.4 Empaquetado y desempaquetado de datos

Hemos visto el principio del empaquetado de datos en un byte; ahora podemos ver cómo se pueden juntar los datos y después cómo se pueden desglosar de nuevo. Generalmente, la unión de datos utiliza instrucciones de desplazamiento e instrucciones OR, y la separación de datos utiliza instrucciones de desplazamiento e instrucciones AND.

La figura 11.6 es un programa para empaquetar datos en la posición de memoria rotulada con ELEM que utiliza las normas dadas en la sección anterior. Al comienzo del programa el registro B contiene la edad real de la persona, el registro C el sexo y el registro D si son de cuota o no.

```
 1Ø REM go
 2Ø REM org 2376Ø
 3Ø REM Elem;def Ø
 35 REM ld a,b
 4Ø REM sub 18;! edad en bits Ø-5
 45 REM !
 5Ø REM ld b,6
 55 REM Sexo;sla c;! mover sexo
    al bit 6
 6Ø REM djnz Sexo
 65 REM or c;!poner sexo en A
 7Ø REM !
 75 REM ld b,7
 8Ø REM Pago;sla d;!mover Pago
    al bit 7
 85 REM djnz Pago
 9Ø REM or d;!poner pago en A
 95 REM !
1ØØ REM ld (Elem),a;!guardar en
    memoria
1Ø5 REM ret
11Ø REM finish
```

Figura 11.6

Esto va seguido por la figura 11.7 que es un programa para desempaquetar los datos. Este invierte el procedimiento del programa anterior.

```
1Ø REM go
2Ø REM org 2376Ø
3Ø REM !Desemp. datos en Elem
35 REM ld a,(Elem)
4Ø REM and $8Ø ;!1ØØØØØØØB
45 REM ld d,a
5Ø REM ld b,7
55 REM Pago1;srl d;!mover a
   bit Ø
6Ø REM djnz Pago1
```

```
 65 REM !
 7Ø REM ld a,(Elem)
 75 REM and $4Ø;!Ø1ØØØØØØB
 8Ø REM ld c,a
 85 REM ld b,6
 9Ø REM Sexol;srl c;!mover a
    bit Ø
 95 REM djnz Sexol
1ØØ REM !
1Ø5 REM ld a,(Elem)
11Ø REM and $3F;!ØØ111111B
115 REM add a,18;!edad correcta
12Ø REM ld b,a
125 REM ret
13Ø REM finish
```

Figura 11.7

11.5 Fichero de atributos

Ya hemos visto anteriormente cómo las imágenes de la televisión se almacenan como puntos (pixels) en el fichero de la pantalla, pero el Spectrum es una computadora de color y hay un área más de memoria que se utiliza para los detalles del color. Este es el fichero de atributos. El fichero de atributos está formado por una posición de memoria para cada posición de carácter de la pantalla y están distribuidos de la forma que espera que lo estén: primero 32 posiciones para la línea superior, después 32 posiciones para la segunda línea y así sucesivamente para las 24 líneas. Cada posición del fichero de atributos está realmente empaquetada con cuatro elementos de datos. Los bits 0 a 2 contienen el código del color de la tinta en binario, los bits 3 a 5 contienen el color del papel, el bit 6 es 1 para brillo y 0 para normal y el bit 7 es 1 para centelleo y 0 para reposo.

La figura 11.8 muestra cómo se puede utilizar el fichero de atributos para modificar los característicos del color de la pantalla.

```
 1Ø REM go
 2Ø REM org 2376Ø
 24 REM equ 768 Bytes
 26 REM 22528 Fich
 3Ø REM !modificar elemento color
    en fichero de atributos
```



```
 35 [illegible] !introducir color de
    tinta
 4Ø REM call Entsal
 45 REM 48;!código a valor
 5Ø REM ld b,a
 55 REM !
 57 REM !introducir color de
    papel
 6Ø REM !call Entsal
 65 REM sub 48
 7Ø REM ld c,a
 75 REM !
 77 REM !introducir brillo
 8Ø REM call Entsal
 85 REM sub 48
 9Ø REM ld d,a
 95 REM !
 98 REM !introducir centelleo
1ØØ REM call Entsal
1Ø5 REM sub 48
11Ø REM ld e,a
115 REM !
12Ø REM !empaquetar datos en A
125 REM xor a;!cero en A
13Ø REM ld a,b;!tinta en bits Ø-2
132 REM !
135 REM ld b,3
14Ø REM Papel; sla c
145 REM djnz Papel
15Ø REM or c;!papel en bits 3-5
155 REM !
16Ø REM ld b,6
165 REM Brillo;sla d
17Ø REM djnz Brillo
175 REM or d;!brillo en bit 6
18Ø REM !
185 REM ld b,7
19Ø REM Cent;sla e
195 REM djnz Cent
2ØØ REM or e;!centelleo en bit 7
2Ø5 REM !
21Ø REM !mover a fich atributo
215 REM ld bc, Bytes
22Ø REM ld hl,Fich
```

```
225 REM push hl
230 REM pop de
235 REM inc de
240 REM ld (hl),a
245 REM ldir;!mover
250 REM ret
255 REM !
260 REM Entsal;call 4264
270 REM cp 208
275 REM jr z,Entsal
280 REM push af
290 REM rst 16
315 REM pop af
320 REM ret
330 REM finish
```

Figura 11.8

11.6 Programa

Escriba una subrutina que visualice en la pantalla el valor contenido en el acumulador en binario. Tendrá que analizar cada bit por separado y visualizar el código de 1 ó 0 dependiendo del valor del bit.

Escriba un programa que pida dos números decimales, comprendidos entre 0 y 255 y un operador lógico, AND, OR o XOR, y que después visualice los dos números en binario y también el resultado de la combinación de acuerdo al operador lógico dado en binario.

12 BLOQUES Y TABLAS

12.1 Búsqueda en bloques

Además de las instrucciones que permiten el movimiento de bloques de memoria veremos ahora aquellas instrucciones que permiten la búsqueda de un valor determinado dentro de un bloque de memoria. Antes de realizar una búsqueda en un bloque el acumulador tiene que contener el valor que está buscando la computadora. La pareja de registros HL contiene la dirección de la primera posición a buscar y la pareja de registros BC contiene el número de posiciones de memoria que pueden buscarse.

La primera instrucción es la CPIR, que indica «comparar, incrementar y repetir». Esta busca a través del bloque de memoria comenzando por la posición indicada por la pareja de registro HL. Después de comparar el valor de cada posición con el acumulador, el valor de la pareja de registros HL se incrementa en uno y el valor de la pareja de registros BC se decrementa en uno. La búsqueda continúa hasta que se produce el encuentro o hasta que se alcance el valor cero en el registro BC. Cuando se produce un encuentro, el indicador de cero se activa (recuerde que al llegar a cero el registro BC no pone el indicador de cero).

Similar a la instrucción anterior es la instrucción CPDR. Que es la instrucción de «comparar, decrementar y repetir» y funciona de la misma forma que la CPIR excepto que ahora la pareja de registros HL tiene que apuntar a la dirección superior del bloque. Cuando la instrucción se ejecuta, se decrementa el valor de la pareja de registros HL. La figura 12.1 es un pequeño programa que busca en un bloque de memoria hasta que encuentra el valor 255 en una de las posiciones.

```
10 REM go
20 REM org 23760
30 REM equ 32600 Bloque
35 REM Buscar;defw 0
40 REM ld hl,Bloque
45 REM ld bc,100;!número de
   posiciones
50 REM ld a,255;!valor buscado
55 REM cpir;! buscar
57 REM !saltar si no se encuentra
```

```
60 REM jr nz,Final
65 REM ld (Buscar),hl;!posici..
   del valor
70 REM Final;ret
75 REM finish
```

Figura 12.1

Finalmente existen dos instrucciones de búsqueda dentro de un bloque que no se repiten de forma automática; éstas son CPI y CPD. Son similares a las dos anteriores excepto en la repetición. Después de cada comparación el programador tiene que escribir unas instrucciones adicionales para comprobar si existe coincidencia entre el acumulador y el valor de memoria; si no es así, se tiene que verificar si se ha llegado al final del bloque. Estas instrucciones se utilizarían en los casos en que se requiere un proceso posterior.

12.2 Registros índice

Hemos visto que con frecuencia se utiliza la pareja de registros HL como un puntero para permitir la utilización directa de los datos contenidos en memoria. Existen otros dos registros de 16 bits que se utilizan como apuntadores de datos en memoria; son los registros IX e IY. A estos registros se les conoce como registros índice y se utilizan con frecuencia junto con los bloques de memoria.

Por desgracia, el Spectrum utiliza el registro IY como puntero del bloque de memoria que contiene las variables del sistema; por tanto, no está disponible para el programador en lenguaje ensamblador.

En general, un registro índice se utiliza para apuntar a la primera posición de un bloque de memoria y el resto de las posiciones del bloque se referencian por su desplazamiento respecto al comienzo del bloque. El segmento de programa de la figura 12.2 muestra cómo se utiliza un registro índice y cómo se refe-

```
 10 REM go
 20 REM org 23760
100 REM equ 32600 Bloque
110 REM !
120 REM ld ix,Bloque
125 REM !
130 REM !
```




```
140 REM set 5,(ix+2);!tercera
    posición
145 REM !
150 REM !
160 REM dec (ix+4);!quinta
    posición
165 REM !
170 REM !
180 REM ld a,(ic+8);!novena
    posición
185 REM !
190 REM !
200 REM finish
```

Figura 12.2

rencian posiciones de memoria determinadas, especificando el registro más un desplazamiento desde el comienzo del bloque.

Los registros índice normalmente se utilizan cuando es necesario hacer referencia a un bloque de datos relacionados como por ejemplo una tabla. El registro IX puede incluso utilizarse aunque no sea necesario referirse a un bloque de datos, dando un desplazamiento cero para referirse a la posición señalada por el registro índice. Un registro índice puede utilizarse en lugar de la pareja HL en cualquier instrucción que utilice el HL como apuntador de una posición de memoria, pero tiene que ir seguido de un desplazamiento.

12.3 Tablas de consulta

Con frecuencia, los datos que están relacionados se almacenan como una tabla o lista. Así se puede posteriormente analizar si contienen un elemento de dato o no. Si se encuentra el elemento de dato en una lista o tabla determinada, se puede utilizar para disparar un salto a una zona nueva del programa. Desde luego, si el dato se encuentra en otra tabla, el salto se hará a otra parte del programa. Un programa que utilice tablas de consulta puede consistir de varias tablas diferentes, cada una con unos saltos asociados, o de una sola tabla con un salto diferente para cada entrada de la tabla. El programa de la figura 12.3 es un ejemplo del primer tipo. Utiliza varias tablas diferentes y al encontrar una entrada en una tabla determinada provoca un salto a una parte del programa. Cada tabla tiene su propio salto asociado.

Aparte de ilustrar la forma de utilizar las tablas de decisión,

```
10 REM go
15 REM org 23760
20 REM !Programa de un solo
   paso
25 REM !
30 REM !espacio para tablas
35 REM Dosoct;defb 6 14 16 22
   24 30 32 38 40 46 48 54 56 63198
   203 206 211 214 219 222 230 238
   246 254
40 REM Tresoct;defb 1 17 33
   34 42 49 50 58 194 195 196 202 2
   04 205 210 212 218 220 226 228 2
   34 242 244 250 252
45 REM Indos;defb 9 25 35 41
   43 57 225 227 229 233
50 REM Indcuatro;defb 33 42 34 203
55 REM Byteec;defb 34 42 67 75
   83 91 115 123
60 REM !Area de ejecución para
   una instrucción
65 REM Posinst;defb 0 0 0 0 201
80 REM Cuentprog;defw 0
85 REM !guardar registros para
   programa MC
90 REM ld hl,0
95 REM push hl
100 REM push hl
105 REM push hl
110 REM push hl
115 REM !dirección comienzo de
   entrada
120 REM call Entnum
125 REM ld (Cuentprog),hl
130 REM ex de, hl;!DE es el
   contador
135 REM !buscar número de bytes/
   instrucción
140 REM Comienzo;ld hl,Posinst
145 REM ld a,(de)
150 REM !Compr. si fin de
   programa MC
155 REM cp 201;! Instrucción RET
160 REM ret z
```



```
165 REM .cargar primer byte
170 REM ld (hl),a
175 REM !¿instrucción índice?
180 REM cp 221;!IX
185 REM jp z,Codigoind
190 REM cp 253;!IY
195 REM jp z,Codigoind
200 REM !"237" instrucción
205 REM cp 237
210 REM jp z,Codigode
215 REM !¿instrucción de 2 bytes?
220 REM ld hl,Dosoct
225 REM ld b,25;!longitud de la
   tabla
230 REM Buclea;cp(hl)
235 REM jp z,Instdos
240 REM inc hl
245 REM djnz buclea
250 REM !instrucción de 3 bytes
255 REM ld hl,Tresoct
260 REM ld b,25;!longitud de la
   tabla
265 REM jp Run
267 REM Bucleb;cp(hl)
270 REM jp z,lnstres
275 REM inc hl
280 REM djnz Bucleb
285 REM !instrucción de un byte
290 REM jp Run
295 REM !instrucciones índice
300 REM !Codigoind,inc de;!segundo
   byte
305 REM ld a,(de)
310 REM ld hl,Indos
315 REM ld b,10;!longitud de la
   tabla
320 REM Buclec;cp(hl)
325 REM jp nz,Próximo
330 REM ld, Posinst+1
335 REM ld (hl),a
340 REM jp Run
345 REM !continuar la búsqueda
350 REM Próximo;inc hl
355 REM djnz Buclec
```

```
360 REM !¿instrucción de 4 bytes?
365 REM ld hl,Indcuatro
370 REM ld b,4;!longitud de la
    tabla
375 REM Bucled;cp(hl)
380 REM jp z,Instcuatro
385 REM inc hl
390 REM djnz Bucled
395 !guardar instrucción de 3 bytes
400 REM ld hl,Posinst+1
405 REM ld (hl),a
410 REM inc de
415 REM inc hl
420 REM ld a,(de)
425 REM ld (hl),a
430 REM jp Run
435 REM !guardar instrucción de 4 bytes
440 REM ld hl,Posinst+1
445 REM ld (hl),a
450 REM inc de
455 REM inc hl
460 REM ld a,(de)
465 REM ld (hl),a
470 REM inc de
475 REM inc hl
480 REM ld a,(de)
485 REM ld (hl),a
490 REM jp Run
495 REM !"237"instrucción
500 REM Codigode;inc de
505 REM ld hl, Byteed
510 REM ld b,8;!longitud de la
    tabla
515 REM ld a,(de)
520 REM Buclee;cp (hl)
525 REM jp z,Instcuatro
530 REM inc hl
535 REM djnz Buclee
540 REM !es una instrucción de 2 bytes
545 REM jp Instdosa
550 REM !instrucción de 2 bytes
555 REM Instdos;inc de
560 REM ld a,(de)
565 REM Instdosa;ld hl,Posinst+1
```



```
570 REM ld (hl),a
575 REM jp Run
580 REM !instrucción de 3 bytes
585 REM Instres; inc de
590 REM ld hl,Posinst+1
595 REM ld a,(de)
600 REM ld (hl),a
605 REM inc de
610 REM inc hl;tercer byte
615 REM ld a,(de)
620 REM ld(hl),a
625 REM jp Run
630 REM !guardar posición próxima
    instrucción

640 REM Run;inc de
650 REM ex de,hl
655 REM ld (Cuentprog),hl
660 REM !restaurar registros
665 REM pop hl
670 REM pop de
675 REM pop bc
680 REM pop af
685 REM !ejecutar la instrucción
690 REM call Posinst
695 REM !visualizar registros
700 REM call Borrarpant
705 REM call Cabeceras
710 REM call Imp
715 REM ld a,b
720 REM call Impa
725 REM ld a,c
730 REM call Imp
735 REM ld a,d
740 REM call Impa
745 REM ld a,e
750 REM call Imp
755 REM ld a,h
760 REM call Impa
765 REM ld a,l
770 REM call Impa
775 REM push af
780 REM pop hl
785 REM ld a,l
```

```
79Ø REM call Impi
795 REM !guardar registros
8ØØ REM push af
8Ø5 REM push bc
81Ø REM push de
815 REM push hl
82Ø REM ld hl,(Cuentprog)
825 REM ex de,hl
83Ø REM ld a,d
835 REM call Impa
84Ø REM ld a,e
845 REM call Imp
85Ø REM !espacio vacío
855 REM ld a,Ø
86Ø REM ld b,4
865 REM ld hl,Posinst
87Ø REM Buclef;ld(hl),a
875 REM inc hl
88Ø REM djnz Buclef
882 REM call Bustecla
885 REM jp Comienzo
89Ø REM !subrutinas
895 REM !
9ØØ REM !borrar pantalla
9Ø5 REM Borrarpant;push af;push bc
91Ø REM push de;push hl
915 REM ld b,24
92Ø REM call 3652
925 REM pop hl;pop de
93Ø REM pop bc;pop af
935 REM ret
94Ø REM !
945 REM !impresión de títulos
95Ø REM Acum;defs Acumulador BCD
    EHL S Z - H - P/V N CFlagsP C
955 REM Cabeceras;ld a,2
96Ø REM call 5633
965 REM ld a,13
97Ø REM ld b,4
975 REM Cabl;ld c,a
98Ø REM rst 16
985 REM ld a,c
99Ø REM djnz Cabl
995 REM !
```



```
1ØØØ REM ld hl,Acumulador
1ØØ5 REM ld b,11
1Ø1Ø REM Cabacum;ld a,(hl)
     !Acumulador
1Ø15 REM rst 16
1Ø2Ø REM inc hl
1Ø25 REM djnz Cabacum
1Ø3Ø REM call Líneasnue.
1Ø35 REM !
1Ø4Ø REM ld b,3
1Ø45 REM Registros;ld a,(hl)
1Ø5Ø REM rst 16
1Ø55 REM inc hl
1Ø6Ø REM ld a,(hl)
1Ø65 REM rst 16
1Ø7Ø REM call Líneasnue.
1Ø75 REM inc hl
1Ø8Ø REM djnz Registros
1Ø85 REM !cabeceras indicadoras
1Ø9Ø REM ld b,7
1Ø95 REM ld a,32
11ØØ REM ld c,a
11Ø5 REM Espacios;rst 16
111Ø REM ld a,c
1115 REM djnz Espacios
112Ø REM !
1125 REM ld b,23
113Ø REM Indicadores;ld a,(hl)
1135 REM rst 16
114Ø REM inc hl
1145 REM djnz Indicadores
115Ø REM ld a,13
1155 REM rst 16
116Ø REM ld b,5
1165 REM Indicador;ld a,(hl)
117Ø REM rst 16
1175 REM inc hl
118Ø REM djnz Indicador
1185 REM call Líneasnue.
119Ø REM ld a,(hl)
1195 REM rst 16
12ØØ REM inc hl
12Ø5 REM ld a,(hl)
121Ø REM rst 16
```

```
1215 REM pop hl;pop de;pop bc;pop
     af
122Ø REM ret
1225 REM !
123Ø REM Líneasnue.;ld a,13;rst 16
1235 REM ld a,13;rst 16
124Ø REM ret
1245 REM !
125Ø REM Impa;push af
1255 REM ld a,6;rst 16
126Ø REM pop af
1263 REM !imprimir valor en
     hexadecimal
1265 REM Imp;push af;push bc;push
     af
127Ø REM and 24Ø
1275 REM ld b,4
128Ø REM Despl;srl a
1285 REM djnz Despl
129Ø REM call Dígito
1295 REM pop af
13ØØ REM and 15
13Ø5 REM call Dígito
131Ø REM pop bc;pop af
1315 REM ret
132Ø REM !
1325 REM !imprimir indicadores
133Ø REM Impi;push af;push bc
1335 REM ld b,8
134Ø REM Inizq;sla a
1345 REM jp nc,Incero
135Ø REM push af
1335 REM ld a,49
136Ø REM jr Inext
1365 REM Incero;push af
137Ø REM ld a,48
1375 REM Inext;rst 16;pop af
138Ø REM djnz Inizq
1385 REM pop bc;pop af
139Ø REM ret
14ØØ REM !esperar pulsación de
     tecla
14Ø5 REM Bustecla;in a.(254)
141Ø REM cpl;jr z,Bustecla;ret
```



```
1415 REM !imprimir dígitos hexadecimales
142Ø REM Dígito;cp 9;jp p,letra
1425 REM add a,48;jr Sald
143Ø REM Letra;add a,55
1435 REM Sald;rst 16;ret
144Ø REM !entrada de número
1445 REM Entnum;ld l,Ø;ld h,Ø
145Ø REM Buclenum;call Entsal
1455 REM cp 13;ret z
146Ø REM call Mul1Ø;sub 48
1465 REM ld d,Ø;ld e,a;add hl,de
147Ø REM jr Buclenum
1475 REM !
148Ø REM Mul1Ø;add hl,hl
1485 REM push hl;pop de;!copiar
     a DE
149Ø REM add hl,hh;add hl,hl
1495 REM add hl,de;ret
15ØØ REM !
15Ø5 REM Entsal;push hl;push de
151Ø REM push bc;Buclent;call 4264
1515 REM cp 2Ø8;jr z,Buclent
152Ø REM push af;rst 16
1525 REM pop af;pop bc;pop de;pop
     hl
153Ø REM ret
1535 REM finish
```

Figura 12.3

este programa constituye un programa de utilidad muy útil para el programador en lenguaje ensamblador o de código máquina. Este es un programa de un solo paso que puede utilizarse para ejecutar otro programa en código máquina instrucción a instrucción. Inicialmente se introduce la dirección de comienzo del programa en código máquina y después se ejecuta una instrucción y se visualizan los valores de los registros. La siguiente instrucción se ejecuta cuando se pulsa cualquier tecla.

12.4 Tablas de salto

Un requerimiento muy usual en programación es comprobar el valor de una variable y saltar a una sección particular del programa dependiendo del valor de la variable.

El resultado es almacenar en memoria una la de instrucciones de saltos, utilizando los registros HL o IX como apuntadores al comienzo de la tabla. El valor de la variable se utiliza para calcular un desplazamiento desde el comienzo de la tabla de saltos. Este valor se suma al valor del registro apuntador para dar la posición dentro de la tabla de saltos. Las instrucciones

JP(HL) o JP(IX)

se utilizan para saltar a la parte requerida del programa. La figura 12.4 muestra la utilización de esta técnica. El programa pide un número de mes del 1 al 12 y partiendo de él visualiza el nombre del mes.

```
 1Ø REM go
 2Ø REM org 2376Ø
 3Ø REM !Introducir número del
    mes, imprime el nombre
 4Ø REM call Enstal
 45 REM sub 48;!código a valor
 5Ø REM ld d,Ø;!poner a Ø
 55 REM ld e,a;!valor a E
 6Ø REM sla a;!2 veces
 65 REM add a,e;!3 veces
 7Ø REM sub 3
 75 REM ld e,a
 8Ø REM ld hl, Tabsal
 85 REM add hl,de
 9Ø REM jp (hl);!saltar a tabla
 95 REM !Tabla de saltos
1ØØ REM Tabsal;jp Ene
1Ø5 REM jp Feb
11Ø REM fb Mar
115 REM jp Abr
12Ø REM jp May
125 REM jp Jun
13Ø REM jp Jul
135 REM jp Ago
14Ø REM jp Sep
145 REM jp Oct
15Ø REM jp Nov
155 REM jp Dic
16Ø REM !
165 REM Ene;ld hl,En
17Ø REM call Texto
```



```
175 REM ret
18Ø REM !
185 REM Feb;ld hl,F
19Ø REM call Texto
195 REM ret
2ØØ REM !
2Ø5 REM Mar;ld hl,Mr
21Ø REM call Texto
215 REM ret
22Ø REM !
225 REM Abr;ld hl,Ab
23Ø REM call Texto
235 REM ret
24Ø REM !
245 REM May;ld hl,My
25Ø REM call Texto
255 REM ret
26Ø REM !
265 REM Jun;ld hl,Jn
27Ø REM call Texto
275 REM ret
28Ø REM !
285 REM Jul;ld hl,Jl
29Ø REM call Texto
295 REM ret
3ØØ REM !
3Ø5 RFM Ago;ld hl,Ag
31Ø REM call Texto
315 REM ret
32Ø REM !
325 REM Sep;ld hl,S
33Ø REM call Texto
335 REM ret
34Ø REM !
345 REM Oct;ld hl,O
35Ø REM call Texto
355 REM ret
36Ø REM !
365 REM Nov;ld hl,N
37Ø REM call Texto
375 REM ret
38Ø REM !
385 REM Dic;ld hl,D
39Ø REM call Texto
```

```
395 REM ret
4ØØ REM !
4Ø5 REM En;defs Enero
41Ø REM defb Ø
415 REM F;defs Febrero
42Ø REM defb Ø
425 REM Mr;defs Marzo
43Ø REM defb Ø
435 REM Ab;defs Abril
44Ø REM defb Ø
445 REM My;defs Mayo
45Ø REM defb Ø
455 REM Jn;defs Junio
46Ø REM defb Ø
465 REM Jl;defs Julio
47Ø REM defb Ø
475 REM Ag;defs Agosto
48Ø REM defb Ø
485 REM S;defs Septiembre
49Ø REM defb Ø
495 REM O;defs Octubre
5ØØ REM defb Ø
5Ø5 REM N;defs Noviembre
51Ø REM defb Ø
515 REM D;defs Diciembre
52Ø REM defb Ø
525 REM !
53Ø REM Entsal;call 4264
535 REM cp 2Ø8;jr z,Entsal
54Ø REM push af;rst 16
555 REM pop af;ret
56Ø REM !
565 REM Texto;ld a,2;call 5633
57Ø REM Rept;ld a,(hl)
575 REM cp Ø
58Ø REM jr z, Final
585 REM rst 16
59Ø REM jr Rept
595 REM Final;ld a,13
6ØØ REM rst 16
6Ø5 REM ret
61Ø REM finish
```

Figura 12.4

Esto, desde luego, es una forma muy sencilla de utilizar la tabla de saltos, puesto que cada segmento de programa tiene la misma longitud y es más fácil de calcular el comienzo de cada segmento de programa.

12.5 Números aleatorios

Para muchos programas de juegos, es esencial poder producir números aleatorios para introducir en el juego el factor necesario de azar, pero la producción de números aleatorios desde un programa ensamblador puede ser una tarea compleja.

La ROM del Spectrum contiene un generador muy bueno de números aleatorios; sin embargo, no es sencillo de utilizar y solamente está recomendado para los programadores experimentados que conozcan bien el programa ROM. Desde luego hay una forma sencilla bajo la cual cualquier programa ensamblador puede utilizar los números aleatorios generados por el programa ROM. Volviendo a BASIC desde su programa, puede utilizar la función RND para producir un número aleatorio, hacer POKE del valor a una posición utilizable de memoria y finalmente volver a su programa ensamblador mediante una llamada USR.

Existe otra forma bajo la cual un programa en lenguaje ensamblador puede producir un número que, aunque no es realmente aleatorio, en casi la totalidad de los casos puede utilizarse como un número aleatorio. Uno de los registros del microprocesador Z80, llamado registro R, lo utiliza el sistema para asegurar que no pierde los datos de memoria; en efecto, esto significa que el valor del registro R está constantemente variando y si se carga su valor en el registro acumulador mediante la instrucción

LD A,R

se cargará el acumulador con un valor comprendido entre 0 y 255 que es razonablemente aleatorio. Si su programa supone entradas desde el teclado, que, desde luego, significa que está utilizando bucles de una longitud indeterminada mientras espera a que se pulse una tecla, este método probablemente proporciona números que son tan aleatorios como el generador de números aleatorios de la ROM.

Aunque al utilizar el registro R se obtiene un número comprendido entre 0 y 255, no es necesario todo este rango. Como ejemplo veamos cómo se puede utilizar este método para simular el lanzamiento de un dado. Necesitamos números aleatorios comprendidos entre 1 y 6, por lo que si tomamos los bits del 0 al 2

de. .número contenido en el registro R tendre[illegible] un número comprendido entre 0 y 7; si ignoramos los ceros y los sietes nos quedamos con un número comprendido en el rango requerido. La figura 12.5 es un programa que simula el lanzamiento de dos dados hasta que se pulse una tecla.

```
 1Ø REM go
 2Ø REM org 2376Ø
 25 REM !primer dado
 3Ø REM Comienzo;ld a,r;!coger
     el número
 4Ø REM and 7;!ØØØØØ111B
 45 REM cp Ø;! Ø no está en el rango
 5Ø REM jp z,Comienzo
 55 REM cp 7;! 7 no está en el
      rango
 6Ø REM add a,48;!valor a código
 7Ø REM ld c,a;!alm. Temp.
 75 REM !segundo dado
 8Ø REM Próximo;ld a,r
 85 REM and 7
 9Ø REM cp Ø
 95 REM jp z, Próximo
1ØØ REM cp 7
1Ø5 REM jp z,Próximo
11Ø REM add a,48
115 REM ld b,a
12Ø REM !Imprimir valores
125 REM ld a,2;call 5633;!abrir
     canal
13Ø REM !Imprimir en 1Ø,1Ø
135 REM ld a,22;!AT
14Ø REM rst 16
145 REM ld a,1Ø;rst 16
15Ø REM ld a,1Ø;rst 16
155 REM ld a,b;rst 16
16Ø REM !Imprimir en 1Ø,13
165 REM ld a,22;rst 16
17Ø REM ld a,1Ø;rst 16
175 REM ld a,13;rst 16
18Ø REM ld a,c;rst 16
185 REM !verificar tecla
19Ø REM in a,(254)
195 REM cpl;jp z,Comienzo
```



```
215 R[illegible] ret
22Ø REM finish
```

Figura 12.5

12.6 Programa

Escriba un programa para el sencillo juego de «Piense un número». La computadora debería producir un número aleatorio entre 0 y 99 y después al jugador se le darán cinco oportunidades para que intente averiguar el número. Después de cada intento el jugador debe recibir un mensaje indicándole si el número elegido era demasiado alto o demasiado bajo.

13 MAS ARITMETICA

13.1 Números de dieciséis bits

Todos los números que hemos utilizado con el cálculo aritmético han sido números de ocho bits, lo que quiere decir que solamente podíamos realizar cálculos aritméticos con un rango muy limitado de números. Sin embargo, el Spectrum tiene varios registros de 16 bits en su procesador central y la utilización de números de 16 bits nos dará un rango mucho más amplio de números, suficientes para la mayoría de los problemas. El microprocesador Z80 incluye algunas instrucciones aritméticas de 16 bits; realmente, estas instrucciones pueden suministrar cálculos aritméticos de 32 bits, 48 bits o incluso números mayores.

Cuando se realizan cálculos aritméticos con 16 bits, la pareja de registros HL se utiliza como acumulador. La instrucción de suma de 16 bits, que tiene el formato:

ADD HL,ss

donde ss es uno de los registros de 16 bits BC, HL o SP, suma el valor contenido en el registro ss al contenido HL y deja el resultado en HL. Así como la instrucción ADD, existe una segunda instrucción para la suma con 16 bits; esta es la instrucción ADC que tiene el mismo formato que la instrucción ADD. La diferencia entre ambas es que la instrucción ADC, además de sumar el valor contenido en el registro ss al valor contenido en HL, también suma el valor del indicador de acarreo al registro HL. Esto significa que se puede sumar un acarreo de una suma previa, por lo que se puede realizar un cálculo aritmético de 32 bits o más. La figura 13.1 es un programa que suma dos números de 32 bits o 4 bytes. Como puede ver, cada número ocupa cuatro posiciones consecutivas de memoria. Se tiene que poner atención al convertir un número de 32 bits contenido en cuatro posiciones consecutivas de memoria a un solo número decimal. Recuerde que 32 bits pueden contener números comprendidos entre 0 y 8 600 000 000. Para convertir un número contenido en cuatro bytes a un solo valor decimal el cálculo es:

Valor total = valor del primer byte *16777216 + valor del segundo byte *65536 + valor del tercer byte *256 + valor del cuarto byte.


```
 1  REM gc
 2Ø REM org 2376Ø
 3Ø REM Primero;defw 5ØØ
 4Ø REM defw 2ØØØ
 5Ø REM Segundo;defw 15Ø
 6Ø REM defw 35Ø
 7Ø REM Resultado:defw Ø
 8Ø REM defw Ø
 9Ø REM !comienzo del programa
1ØØ REM ld hl,(Primero+1)
11Ø REM ld bc,(Segundo+1)
12Ø REM add hl,bc
13Ø REM ld (Resultado+1),hl
14Ø REM ld hl,(Primero)
15Ø REM ld bc,(Segundo)
16Ø REM adc hl,bc
17Ø REM ld (Resultado),hl
18Ø REM ret
19Ø REM finish
```

Figura 13.1

Aunque hay dos instrucciones diferentes para la suma de números de 16 bits, solamente existe una instrucción de resta. Es la instrucción SBC y su formato es:

SBC HL,ss

donde ss es uno de los registros de 16 bits BC, DE, HL o SP. El efecto producido por la instrucción SBC es restar el valor del registro de 16 bits y el valor del indicador de acarreo del valor contenido en el registro HL, dejando el resultado en el registro HL.

La instrucción SBC puede utilizarse para realizar sustracciones de 32 bits, pero antes de que se puedan restar números de 16 bits o los 16 primeros bits de un número de 32 bits se tiene que asegurar que el valor del indicador de acarreo es cero.

Las dos instrucciones que permiten modificar directamente el indicador de acarreo son, como debería recordar:

SCF

que pone el valor del indicador de acarreo a uno y

CCF

que cambia el valor del indicador de acarreo al opuesto del valor actual. La figura 13.2 es un programa que lleva a cabo una resta de 32 bits.

```
 1Ø REM go
 2Ø REM org 2376Ø
 3Ø REM Primero;defw 5ØØ
 4Ø REM defw 2ØØØ
 5Ø REM Segundo;defw 15Ø
 6Ø REM defw 35Ø
 7Ø REM Resultado;defw Ø
 8Ø REM defw Ø
 9Ø REM !comienzo del programa
1ØØ REM ld hl,(Primero+1)
11Ø REM ld bc,(Segundo+1)
114 REM scf
117 REM ccf
12Ø REM sbc hl,bc
13Ø REM ld (Resultado+1),hl
14Ø REM ld hl,(Primero)
15Ø REM ld bc,(Segundo)
16Ø REM sbc hl,bc
17Ø REM ld (Resultado),hl
18Ø REM ret
19Ø REM finish
```

Figura 13.2

Una observación final, aunque las instrucciones ADC y SBC afectan a los indicadores de acarreo, desbordamiento, signo y cero, como debería imaginar la instrucción ADD de 16 bits no lo hace, solamente afecta al indicador de acarreo. Esto quiere decir que incluso cuando se utilice la instrucción de ADD de 16 bits normalmente primero se comprobará que el indicador de acarreo contenga el valor cero.

13.2 Números múltiplos de byte

Acabamos de ver cómo las instrucciones ADC y SBC permiten sumar y restar números que sean múltiplos de 16 bits. Hay, sin embargo, versiones de ocho bits de estas instrucciones que permiten sumar o restar números que sean múltiplos de ocho bits. El formato de las instrucciones es:

ADC A,s
SBC A,s

donde s es un valor de ocho bits, un registro de ocho bits o un valor en una posición de memoria señalada por la pareja de registros HL.

La figura 1[illegible] es un programa que lleva a cabo una suma [illegible]tiplo de byte. En este programa el registro B contiene el número de bytes de los números.

```
 1Ø REM go
 2Ø REM org 2376Ø
 3Ø REM !áreas de datos
 35 REM Long;defb 3
 4Ø REM Prim;defb 12
 45 REM defb 3Ø
 5Ø REM defb 1ØØ
 55 REM Segun;defb 5
 6Ø REM defb 78
 65 REM defb 195
 7Ø REM Result;defb Ø
 75 REM defb Ø
 8Ø REM defb Ø
 85 REM !iniciar acarreo
 9Ø REM scf
 95 REM ccf
1ØØ REM !número de bytes en B
1Ø5 REM ld a,(Long)
11Ø REM ld b,a
115 REM !apuntar a los primeros bytes
12Ø REM ld de,(Prim+2)
12Ø REM ld de,(Prim+2)
125 REM ld hl,(Segun+2)
13Ø REM ld ix,(Result+2)
135 REM !realizar suma
14Ø REM Bucle;ld a,(de)
145 REM adc a,(hl)
15Ø REM ld (ix+Ø),a
155 REM dec de
16Ø REM dec hl
165 REM dec ix
17Ø REM djnz Bucle
175 REM ret
18Ø REM finish
```

Figura 13.3

13.3 Decimal codificado en binario

Siempre que hemos utilizado números, incluso aunque se hayan introducido por el teclado como números decimales, hemos utilizado su representación binaria. Ya habrá comprobado lo difícil que resulta convertir de binario a dígitos binarios y después a códigos de carácter para la salida. Afortunadamente, se puede utilizar otra representación; la decimal codificada en binario o BCD. En esta representación cada dígito del número decimal se representa de forma independiente. Cada dígito se expresa como un número binario de cuatro bits. A cada grupo de cuatro bits se le conoce como nibble (cuaterna) y dos nibbles forman un byte; cualquier registro de ocho bits o cualquier posición de memoria puede contener dos dígitos de un número BCD. La figura 13.4 muestra cómo se puede almacenar un número decimal de cuatro dígitos en dos posiciones sucesivas de memoria.

Memoria en decimal	Direcciones 7438 en BCD	Memoria en binario
	32513	
3 ¦ 8	32512	00111000
7 ¦ 4	32511	01110100
	32510	

Figura 13.4

Cuando se utiliza un nibble para contener un dígito decimal, contiene un número comprendido entre 0 y 9, pero si se utiliza la representación binaria para cuatro bits pueden representar números comprendidos entre 0 y 15; por tanto, la representación BCD desaprovecha espacio.

Otra desventaja de los números BCD se presenta al realizar operaciones aritméticas. La computadora está, desde luego, esperando números binarios puros y proporciona resultados erróneos con el BCD; esto se discute con detalle en la próxima sección. Para utilizar números BCD tenemos que ser capaces de mover los dígitos desde memoria al acumulador y viceversa. Desde luego, podríamos mover dos dígitos de números BCD a la vez, o podríamos utilizar las instrucciones de desplazamiento o rotación para mover un bit cada vez, pero ambos métodos son incómodos. Hay dos instrucciones que proporcionan un movimiento directo entre memoria y el acumulador mediante nibbles de datos. Estas instrucciones son realmente rotaciones que utilizan el nibble de la parte derecha del acumulador y los dos nibbles de una posición de memoria. Las dos instrucciones son:

RLD   Rotar el dígito de la izquierda.
RRD   Rotar el dígito de la derecha.

Antes de ejecutar cualquiera de estas dos instrucciones, se utiliza el registro HL para apuntar a la posición de memoria requerida. La figura 13.5 muestra la forma de operar de las instrucciones.

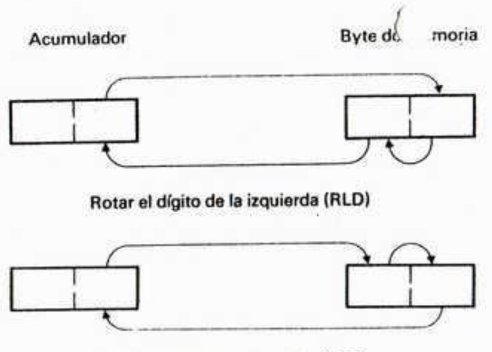
13.4 Aritmética BCD

Si tomamos algunos números BCD de dos dígitos y los sumamos utilizando la aritmética binaria y después interpretamos los resultados como números BCD, vemos que algunas veces el resultado es correcto y otras no:

```
34—00110100
51—01010001
   10000101—85 Respuesta correcta
37—00110111
59—01011001
   10010000—90 Respuesta errónea
36—00110110
55—01010101
   10001011—8?
```

Como puede observar en la última respuesta, algunas veces no se puede ni siquiera representar el resultado como un dígito binario ya que los cuatro bits representan un número superior a nueve.

Existen dos métodos utilizados por las computadoras para asegurar la fiabilidad de los cálculos aritméticos BCD. Un método es suministrar un conjunto de instrucciones completamente independientes para la aritmética BCD; y otro que proporciona una forma de corregir los resultados de los cálculos aritméticos binarios para los números BCD. El procesador central del Spectrum utiliza este segundo método. Cuando se utilizan números BCD,



Rotar el dígito de la izquierda (RLD)

Rotar el dígito de la derecha (RRD)

Figura 13.5

las instrucciones aritméticas (ADD, ADC, SUB o SBC) deberían ir seguidas inmediatamente por la instrucción:

DAA

Esta instrucción corrige cualquier error producido debido al uso de la aritmética binaria y proporciona la respuesta BCD correcta. El programa de la figura 13.6 muestra la entrada, suma y salida de números BCD de dos dígitos.

```
 10 REM go
 20 REM org 23760
 30 REM !suma BCD de 4 dígitos
 35 REM Número1;defw 0
 40 REM Número2;defw 0
 45 REM Result;defw 0
 50 REM !entrada primer número
 55 REM 1d h1,Número1
 60 REM call Dosdig
 65 REM inc h1
 70 REM call Dosdig
 75 REM !entrada segundo número
 80 REM inc h1
 85 REM call Dosdig
 90 REM inc h1
 95 REM call Dosdig
100 REM !suma
105 REM 1d h1,Número1
110 REM 1d a,(h1)
120 REM 1d h1,Número2
130 REM add a,(h1)
```



```
135 REM daa; !corregir para BCD
140 REM 1d h1,Result
145 REM 1d (h1),a
150 REM segunda pareja de dígitos
155 REM 1d h1,Número1+1
165 REM 1d a,(h1)
170 REM 1d h1,Número2+1
175 REM adc a,(h1)
180 REM daa
185 REM 1d h1,Result+1
190 REM 1d (h1),a
193 REM ret
195 REM !subrutina de entradas
200 REM Dosdig;call Entsal
205 REM sub 48
210 REM 1d (h1),a
215 REM call Entsal
220 REM sub 48
225 REM rld
230 REM ret
235 REM Entsal;call 4264
240 REM cp 208;jr z,Entsal
245 REM push af;rst 16
260 REM pop af'ret
270 REM finish
```

Figura 13.6

También se utiliza la instrucción BCD para proporcionar la respuesta BCD correcta después de las instrucciones INC, DEC, CP y NEG. La instrucción DAA solamente actúa sobre el acumulador.

13.5 Otras instrucciones

Hemos visto la mayoría de las instrucciones más prácticas del lenguaje ensamblador, pero esta sección revisará unas pocas instrucciones más que podrían ser útiles.

Dentro del procesador central del Spectrum, existe otro conjunto de registros de ocho bits, llamados registros auxiliares. Estos registros tienen los mismos nombres que los registros de ocho bits principales y se pueden utilizar de la misma forma que éstos, pero no al mismo tiempo.

Los dos conjuntos de registros se pueden intercambiar mediante la instrucción:

EXX

E instrucción permuta los dos conjuntos de re :ros y después de su ejecución todas las instrucciones se referían al segundo conjunto de registros. Al ejecutar de nuevo la instrucción se permutan de nuevo los registros, recuperando el primer conjunto de registros. En vez de intercambiar el conjunto de registros, se puede cambiar solamente el acumulador y los registros indicadores mediante la instrucción:

```
EX AF,AF'
```

Hemos visto anteriormente cómo el procesador central puede enviar datos al mundo exterior mediante la instrucción OUT. De forma similar, el procesador central puede recibir datos del mundo exterior mediante la instrucción IN. Ambas instrucciones tienen dos formatos. El primero es:

```
IN A,(n)
OUT A,(n)
```

donde A es el acumulador y n es el número de port. El número de port es un número de 16 bits. El segundo formato de la instrucción es:

```
IN r,(C)
OUT r,(C)
```

donde r es uno de los registros de ocho bits y el número del port está en la pareja de registros BC.

Una instrucción que en principio parece tener poca o ninguna utilidad para el programador es la instrucción NOP. NOP significa «no operación» y es una instrucción para no hacer nada. Los programadores experimentados la encuentran muy útil. Tiene dos aplicaciones principales: aunque no hace nada consume una determinada cantidad de tiempo en ejecutarse y se utiliza para afinar los tiempos de los bucles.

Después de escribir y ensamblar un gran programa, es muy posible que quiera hacerle alguna modificación. Todo el mundo modifica sus programas. Como el programa en código máquina está almacenado en posiciones sucesivas de memoria, el insertar instrucciones extra puede ser dificultoso. Una técnica muy útil es separar las secciones de un programa con varias instrucciones NOP; después se pueden insertar fácilmente instrucciones extra o saltos a nuevas secciones de código.

Finalmente, en esta sección, la última instrucción que veremos es la instrucción HALT. En muchos aspectos es muy similar a la instrucción STOP de BASIC, pero en el Spectrum el programa solamente se detiene hasta que recibe una señal llamada interrupción que s nvía después de producirse cada imagen e televisión.

Esta sección ha completado las instrucciones del lenguaje ensamblador que hemos visto con detalle. Existen todavía unas pocas instrucciones que no hemos tratado, pero que se utilizan con tan poca frecuencia que la mayoría pueden ignorar aunque no los programadores experimentados. Debería ser capaz ahora de escribir programas importantes en lenguaje ensamblador. Todas las instrucciones que se pueden utilizar con el Spectrum se muestran en el Apéndice A.

13.6 Programa

Escriba un programa que introduzca, sume o reste y después visualice el resultado de números BCD de hasta 6 dígitos.

La entrada consistirá en un número decimal con o sin un signo negativo y un signo más o menos seguido de otro número decimal. La salida debería ser de la siguiente forma:

```
 123+456=579
 772+-123 =649
-123-456  =-579
```

14 ORDENACION DE DATOS

14.1 Clasificación de datos

Me gustaría utilizar este último capítulo para analizar dos métodos que pueden utilizarse para clasificar datos en un determinado orden. Hay por lo menos 40 métodos diferentes para clasificar datos, pero para la mayoría de los programas en lenguaje ensamblador solamente es necesario elegir entre dos métodos diferentes. Los dos métodos son: uno sencillo de programar, que es relativamente lento, y otro más avanzado que es más complejo de programar pero más rápido.

Probablemente lo mejor sería decir que el mejor método de clasificar datos en la memoria de la computadora es ponerles en el orden adecuado cuando se introducen en la computadora. Con las instrucciones tan rápidas como el movimiento de un bloque y la búsqueda en un bloque, se pueden colocar los elementos de datos en el lugar adecuado según se van introduciendo.

14.2 Clasificación burbuja

La clasificación más sencilla y más simple es la *clasificación burbuja.* El principio básico de la clasificación burbuja es comparar elementos de la lista; si los elementos están en orden erróneo se intercambian y después se comparan la siguiente pareja de la lista. Este proceso se repite hasta que todos los elementos estén en el orden correcto. La figura 14.1 es un diagrama de flujo para una clasificación burbuja y la figura 14.2 es un programa que lleva a cabo una clasificación burbuja de una lista de datos en un bloque de memoria. La clasificación burbuja es muy sencilla de programar y es relativamente lenta, pero en lenguaje ensamblador debería ser suficiente para la mayoría de las aplicaciones.

14.3 Clasificación cubierta

La clasificación cubierta es en muchos aspectos una versión mejorada de la clasificación burbuja, pero es significativamente más rápida y es más práctica cuando se requiere una mayor velocidad.

La clasificación cubierta, como la clasificación burbuja, reali-

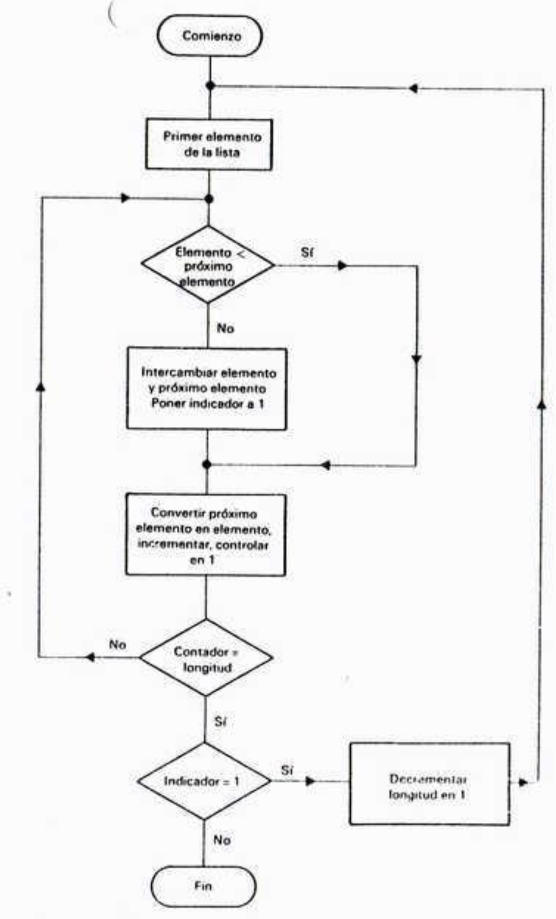


Figura 14.1

za varios pasos sobre los datos para ordenarlos, pero, al contrario que la clasificación burbuja, no compara elementos de datos adyacentes. En el primer paso compara el primer elemento con el elemento del medio de la lista y después compara el segundo con el siguiente al del medio y así hasta alcanzar el final de la lista. En el siguiente paso se divide por dos la distancia de elementos a comparar por lo que ahora se comparará con el que se encuentra en la cuarta parte de la lista. Al final de cada paso la distancia en-

```
10 REM go
20 REM org 23760
30 REM !Clasificación burbuja
35 REM Temp;defb Ø
40 REM ld e,Ø;!indicador para
   mostrar intercambios
41 REM !B contiene la longitud
   de la lista
43 REM !hl apunta al comienzo
   de la lista
45 REM Buclext;ld a,(hl)
50 REM inc hl;!próximo elemento
55 REM ld c,2;!contador de
   elementos
60 REM Bucleint;cp(hl);!comprobar
   orden
65 REM jp m,Próximo
70 REM !intercambiar elementos
75 REM ld d,a
80 REM ld a,(hl)
85 REM dec hl
90 REM ld (hl),a
95 REM inc hl
100 REM ld (hl),d
105 REM ld e,1;!poner indicador
110 REM Próximo;ld a,(hl)
115 REM inc hl
120 REM inc c
125 REM ld d,a
130 REM ld a,b
135 REM sub c;!comprobar fin de
   la lista
140 REM jr nz,Bucleint
145 REM ld a,e
150 REM cp Ø;!si Ø lista
   clasificada
155 REM ret z
160 REM dec b;!reducir longitud
   en 1
165 REM jp Buclext
170 REM finish
```

Figura 14.2

tre los elementos a comparar se divide por dos hasta que la lista queda clasificada. La figura 14.3 muestra las comparaciones realizadas en los tres primeros pasos de una lista de 26 elementos. Las figuras 14.4 y 14.5 son el diagrama de flujo y el programa de una clasificación cubierta.

Longitud de la lista = 26 elementos

Primer paso	Segundo paso	Tercer paso	Cuarto paso
Diferencia 13	Diferencia 6	Diferencia 3	Diferencia 1
1 – 14	1 – 7	1 – 4	1 – 2
2 – 15	2 – 8	2 – 5	2 – 3
3 – 16	.	.	3 – 4
.	.	.	.
.	.	.	.
.	7 – 13	5 – 8	.
12 – 25	8 – 14	6 – 9	24 – 25
13 – 26	.	.	25 – 26
	.	.	
	.	.	
	19 – 25	22 – 25	
	20 – 26	23 – 26	

Figura 14.3

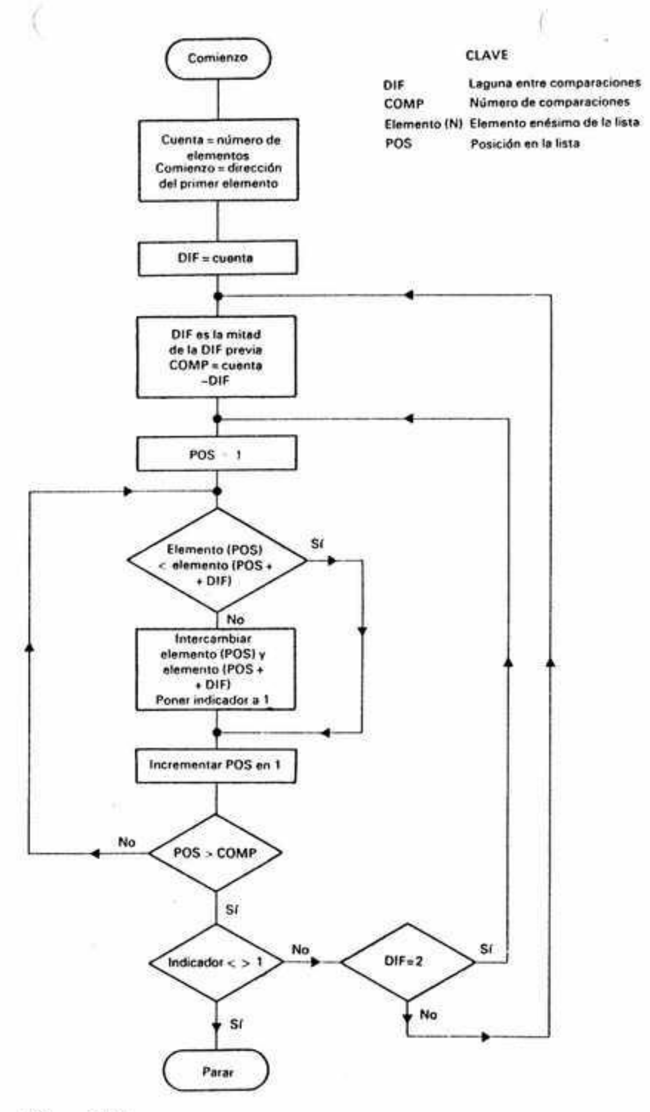


Figura 14.4


```
   REM go
20 REM org 23760
30 REM !Programa clasificación
   cubierta
31 REM !hasta para 255
   elementos
35 REM Cuenta;defbØ
4Ø REM Indicador;defbØ
45 REM !HL apunta al primer elemento
5Ø REM ld d,Ø
55 REM !B contiene el número
   de elementos
6Ø REM ld a,b
62 REM ld (Cuenta),a
65 REM ld e,a
7Ø REM !buscar diferencia
75 REM Buclem;sra e;!dividir
   por 2
8Ø REM sub e
85 REM ld b,a;número de
   comparaciones
87 REM xor a;!Ø en A
88 REM ld (Indicador),a
9Ø REM Buclei;ld a,(hl);!primer
   número
95 REM add hl,de
1ØØ REM cp (hl);!segundo número
1Ø5 REM jp m,Próximo
11Ø REM !intercambiar elementos
115 REM ld c,(hl)
12Ø REM ld (hl),a
125 REM and a;!Ø al indicador
    de acarreo
13Ø REM sbc hl,de
135 REM ld (hl),c
14Ø REM ld a,1
145 REM ld (Indicador),a;!poner
    indicador de intercambio
15Ø REM jr Próx1
155 REM Próximo;and a
16Ø REM sbc hl,de
165 REM Próx1;inc hl
17Ø REM djnz Buclei
175 REM ld a,(Indicador)
```

```
18Ø REM cp Ø
185 REM ret z
19Ø REM ld a,e
195 REM cp 2
2ØØ REM jp nz,Buclem
2Ø5 REM xor a
21Ø REM ld (Indicador),a
215 REM ld a,(Cuenta)
22Ø REM dec a
225 REM ld b,a
23Ø REM ld (Cuenta),a
235 REM jp Buclei
24Ø REM finish
```

Figura 14.5

APENDICE A

RESUMEN DE LAS INSTRUCCIONES DEL LENGUAJE ENSAMBLADOR

Este apéndice contiene todas las instrucciones reconocidas por el microprocesador utilizado en el Spectrum.

La tabla A.1 resume los efectos producidos por las instrucciones en los bits del registro indicador. Solamente se muestran aquellas que afectan a los indicadores. El resto de las tablas muestran todas las instrucciones y su código máquina equivalente en decimal.

Tabla A.1. Registro indicador

	Indicadores					
Instrucción	C	Z	P/V	S	N	H
ADD A	*	*	V	*	0	*
ADC A	*	*	V	*	0	*
SUB	*	*	V	*	1	*
SBC A	*	*	V	*	1	*
CP	*	*	V	*	1	*
NEG	*	*	V	*	1	*
AND	0	*	P	*	0	1
OR	0	*	P	*	0	1
XOR	0	*	P	*	0	0
INC m	–	*	V	*	0	*
DEC m	–	*	V	*	1	*
ADD HL	*	–	–	–	0	–
ADC HL	*	*	V	*	0	–
SBC HL	*	*	V	*	1	–
RLA, RLCA	*	–	–	–	0	0
RRA, RRCA	*	–	–	–	0	0
Rotaciones y desplazamientos	*	*	P	*	0	0
RLD, RRD	–	*	P	*	0	0
DAA	*	*	P	*	–	*
CPL	–	–	–	–	1	1
SCF	1	–	–	–	0	0

Instrucción	C	Z	Indicadores P/V	S	N	H
CCF	*	–	–	–	0	–
IN	–	*	P	*	0	0
INI, IND, OUTI, OUTD	–	*	–	–	1	–
UNIR, INDR, OTIR, OTDR	–	1	–	–	1	–
LDI, LDD	–	–	*	–	0	0
LDIR, LDDR	–	–	0	–	0	0
CPI, CPIR, CPD, CPDR	–	*	*	*	1	–
BIT	–	*	–	–	0	1
NEG	*	*	V	*	1	*

En la tabla se utiliza la notación siguiente:

- m Solamente operandos de ocho bits
- * Indicador afectado
- – Indicador no afectado
- 0 Indicador a 0
- 1 Indicador a 1
- V El indicador muestra desbordamiento
- P El indicador muestra paridad

Tabla A Instrucciones de carga de ocho bits—LD d,s

Destino	Fuente de los datos A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	n
A	127	120	121	122	123	124	125	126	221 126 d	253 126 d	63 n
B	71	64	65	66	67	68	69	70	221 70 d	253 70 d	6 n
C	79	72	73	74	75	76	77	78	221 78 d	253 78 d	14 n
D	87	80	81	82	83	84	85	86	221 86 d	253 86 d	22 n
E	95	88	89	90	91	92	93	94	221 94 d	253 94 d	30 n
H	103	96	97	98	99	100	101	102	221 102 d	253 102 d	38 n
L	111	104	105	106	107	108	109	110	221 110 d	253 110 d	46 n
(HL)	119	112	113	114	115	116	117				54 n
(IX+d)	221 119 d	221 112 d	221 113 d	221 114 d	221 115 d	221 116 d	221 117 d				
(IY+d)	253 119 d	253 112 d	253 113 d	253 114 d	253 115 d	253 116 d	253 117 d				

Destino	Fuente de los datos (BC)	(DE)	(nn)	R	I	A
A	10	26	58 n n	237 95	237 87	
(BC)						2
(DE)						18
(nn)						50 n n
R						237 79
I						237 71

Tabla A.3. Instrucciones de carga de dieciséis bits

	Fuente de los datos nn	(nn)
BC	1, n, n	237, 75, n, n
DE	17, n, n	237, 91, n, n
HL	33, n, n	42, n, n
SPP	49, n, n	237, 123, n, n
IX	221, 33, n, n	221, 42, n, n
IY	253, 33, n, n	253, 42, n, n

	Fuente de los datos BC	DE	HL	SP	IX	IY
(nn)	237 67 n n	237 83 n n	34 n n	237 115 n n	221 34 n n	253 34 n n

Tabla A.4. Instrucciones PUSH y POP

	AF	BC	DE	HL	XI	IY
PUSH	245	197	213	229	221 229	253 229
POP	241	193	209	225	221 225	253 225

Tabla A.5
Instrucciones de intercambio

EXX	217
EX AF,AF'	8
EX, DE,HL	235
EX (SP),HL	227
EX (SP),IX	221, 227
EX (SP),IY	253, 227

Tabla A.6
Instrucciones de bloque

LDI	237, 160
LDIR	237, 176
LDD	237, 168
LDDR	237, 184
CPI	237, 161
CPIR	237, 177
CPD	237, 169
CPDR	237, 185

Tabla A.7
Aritmética general

DAA	39
CPL	47
NEG	237,68
CCF	63
SCF	55

Tabla A.8. Lógica y aritmética de ocho bits

	Fuente de los datos										
	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	n
ADD	135	128	129	130	131	132	133	134	221, 134, d	253, 134, d	198, n
ADC	143	136	137	138	139	140	141	142	221, 142, d	253, 142, d	206, n
SUB	151	144	145	146	147	148	149	150	221, 150, d	253, 150, d	214, n
SBC	159	152	153	154	155	156	157	158	221, 158, d	253, 158, d	222, n
AND	167	160	161	162	163	164	165	166	221, 166, d	253, 166, d	230, n
XOR	175	168	169	170	171	172	173	174	221, 174, d	253, 174, d	238, n
OR	183	176	177	178	179	180	181	182	221, 182, d	253, 182, d	246, n
CP	191	184	185	186	187	188	189	190	221, 190, d	253, 190, d	254, n
INC	60	4	12	20	28	36	44	52	221, 52, d	253, 52, d	
DEC	61	5	13	21	29	37	45	53	221, 53, d	253, 53, d	

Tabla A.9. Aritmética de dieciséis bits

	Fuente de los datos					
	BC	DE	HL	SP	IX	IY
ADD HL	9	25	41	57		
ADD IX	221 9	221 25		221 57	221 41	
ADD IY	253 9	253 25		253 57		253 41
ADC	237 74	237 90	237 106	237 122		
SBC	237 66	237 82	237 98	237 114		
INC	3	19	35	51	221 35	253 35
DEC	11	27	43	59	221 43	253 43

Tabla A[illegible]. Instrucciones de rotación y desplazamiento

	Fuente de los datos									
	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
RLC(A)	7	203 0	203 1	203 2	203 3	203 4	203 5	203 6	221 203 d 6	253 203 d 6
RRC(A)	15	203 8	203 9	203 10	203 11	203 12	203 13	203 14	221 203 d 14	253 203 d 14
RL(A)	23	203 16	203 17	203 18	203 19	203 20	203 21	203 22	221 203 d 22	253 203 d 22
RR(A)	31	203 24	203 25	203 26	203 27	203 28	203 29	203 30	221 203 d 30	253 203 d 30
SLA	203 39	203 32	203 33	203 34	203 35	203 36	203 37	203 38	221 203 d 38	253 203 d 38
SRA	203 47	203 40	203 41	203 42	203 43	203 44	203 45	203 46	221 203 d 46	253 203 d 46
SRL	203 63	203 56	203 57	203 58	203 59	203 60	203 61	203 62	221 203 d 62	253 203 d 62
RLD								237 111		
RRD								237 103		

Tabla A.11. Instrucciones de bit

La instrucción BIT

Número del bit	0	1	2	3	4	5	6	7
	203	203	203	203	203	203	203	203
A	71	79	87	95	103	111	119	127
	203	203	203	203	203	203	203	203
B	64	72	80	88	96	104	112	120
	203	203	203	203	203	203	203	203
C	65	73	81	89	97	105	113	121
	203	203	203	203	203	203	203	203
D	66	74	82	90	98	106	114	122
	203	203	203	203	203	203	203	203
E	67	75	83	91	99	107	115	123
	203	203	203	203	203	203	203	203
H	68	76	84	92	100	108	116	124
	203	203	203	203	203	203	203	203
L	69	77	85	93	101	109	117	125
	203	203	203	203	203	203	203	203
(HL)	70	78	86	94	102	110	118	126
	221	221	221	221	221	221	221	221
(IX+d)	203	203	203	203	203	203	203	203
	d	d	d	d	d	d	d	d
	70	78	86	94	102	110	118	126
	253	253	253	253	253	253	253	253
(IY+d)	203	203	203	203	203	203	203	203
	d	d	d	d	d	d	d	d
	70	78	86	94	102	110	118	126

La instrucción RES

Número del bit	0	1	2	3	4	5	6	7
	203	203	203	203	203	203	203	203
A	135	143	151	159	167	175	183	191
	203	203	203	203	203	203	203	203
B	128	136	144	152	160	168	176	184
	203	203	203	203	203	203	203	203
C	129	137	145	153	161	169	177	185
	203	203	203	203	203	203	203	203

La instrucción RES

Número del bit	0	1	2	3	4	5	6	7
D	130	138	146	154	162	170	178	186
	203	203	203	203	203	203	203	203
E	131	139	147	155	163	171	179	187
	203	203	203	203	203	203	203	203
H	132	140	148	156	164	172	180	188
	203	203	203	203	203	203	203	203
L	133	141	149	157	165	173	181	189
	203	203	203	203	203	203	203	203
(HL)	134	142	150	158	166	174	182	190
	221	221	221	221	221	221	221	221
(IH+d)	203	203	203	203	203	203	203	203
	d	d	d	d	d	d	d	d
	134	142	150	158	166	174	182	190
	253	253	253	253	253	253	253	253
(IH+d)	203	203	203	203	203	203	203	203
	d	d	d	d	d	d	d	d
	134	142	150	158	166	174	182	190

La instrucción SET

Número del bit	0	1	2	3	4	5	6	7
	203	203	203	203	203	203	203	203
A	199	207	215	223	231	239	247	255
	203	203	203	203	203	203	203	203
B	192	200	208	216	224	232	240	248
	203	203	203	203	203	203	203	203
C	193	201	209	217	225	233	241	249
	203	203	203	203	203	203	203	203
D	194	202	210	218	226	234	242	250
	203	203	203	203	203	203	203	203
E	195	203	211	219	227	235	243	251
	203	203	203	203	203	203	203	203
H	196	204	212	220	228	236	244	252

La instrucción SET

Número del bit	0	1	2	3	4	5	6	7
	203	203	203	203	203	203	203	203
L	197	205	213	221	229	237	245	253
	203	203	203	203	203	203	203	203
(HL)	198	206	214	222	230	238	246	254
	221	221	221	221	221	221	221	221
(IX+d)	203	203	203	203	203	203	203	203
	d	d	d	d	d	d	d	d
	198	206	214	222	230	238	246	254
	253	253	253	253	253	253	253	253
(IY+d)	203	203	203	203	203	203	203	203
	d	d	d	d	d	d	d	d
	198	206	214	222	230	238	246	254

Tabla A.12. Instrucciones de salto, llamada y retorno

	Condición								
Instrucción	Ning.	C	NC	Z	NZ	PE	PO	M	P
JP nn	195	218	210	202	194	234	226	250	242
	n	n	n	n	n	n	n	n	n
	n	n	n	n	n	n	n	n	n
JR n	24	56	48	40	32				
	n	n	n	n	n				
JP (HL)	233								
JP (IX)	221								
	233								
JP (IY)	253								
	233								
CALL nn	205	220	212	204	196	236	228	252	244
	n	n	n	n	n	n	n	n	n
	n	n	n	n	n	n	n	n	n
RET	201	216	208	200	192	232	224	248	240
DJNZ	16								
	n								
RETI	237								
	77								
RETN	237								
	69								

Tabla A.13
Instrucciones de reinicialización

RST 0	199
RST 8	207
RST 16	215
RST 24	223
RST 32	231
RST 40	239
RST 48	247
RST 56	255

Tabla A.14. Instrucciones de entrada y salida

		Registro						
Instrucción	A,(n)	A,(C)	B,(C)	C,(C)	D,(C)	E,(C)	H,(C)	L,(C)
IN	219	237	237	237	237	237	237	237
	n	120	64	72	80	88	96	104
OUT	211	237	237	237	237	237	237	237
	n	121	65	73	81	89	97	105
INI	237,162							
INIR	237,178							
IND	237,170							
INDR	237,186							
OUTI	237,163							
OTIR	237,179							
OUTD	237,171							
OTDR	237,187							

Tabla A.15
Instrucciones varias

NOP	0
HALT	118
DI	243
EI	251
IMO	237,70
IM1	237,86
IM2	237,94

APENDICE B
ENSAMBLADOR DE CODIGO MAQUINA DEL ZX SPECTRUM

B.1 Utilización de un ensamblador

Todos los programas de este libro se han producido utilizando el programa Ensamblador en Código Máquina del ZX Spectrum del Software ACS. Este apéndice describe la utilización de este programa ensamblador, pero las ideas generalmente se pueden aplicar a cualquier programa ensamblador del Spectrum.

El programa ensamblador en sí es un programa en código máquina que se carga en la parte superior de la memoria mediante un programa BASIC asociado. Una vez cargado el programa en código máquina se elimina de forma automática el programa BASIC.

Cuando sea necesario, se puede llamar al programa ensamblador mediante la orden BASIC, RANDOMIZE USR 26 000 (para el Spectrum de 16K) o RANDOMIZE USR 58 000 (para el Spectrum de 48K).

Ya hemos visto que se utilizan dos áreas de memoria cuando se traducen los programas escritos en lenguaje ensamblador. Un área se utiliza para almacenar el programa en lenguaje ensamblador y el otro para el programa en código máquina. Durante el desarrollo de un programa es más sencillo reservar un área de memoria en una sentencia REM, al comienzo del programa BASIC, para el programa en código máquina. Casi todos los programas que hemos mostrado en este libro se han desarrollado utilizando este método. El programa BASIC asociado tiene que comenzar con una sentencia REM que contenga tantos caracteres como bytes de memoria ocupados por el programa en código máquina. Normalmente no conocerá el número de bytes requeridos antes de que haya traducido su programa en lenguaje ensamblador. Se puede hacer una estimación admitiendo dos bytes por cada instrucción en lenguaje ensamblador.

B.2 Ensamblador del ZX Spectrum

El Ensamblador en Código Máquina del ZX Spectrum también utiliza el área de programa BASIC para almacenar las ins-

t..ciones en lenguaje ensamblador. Todas las .strucciones se escriben en un programa BASIC dentro de sentencias REM. Un vistazo a alguno de los programas del libro mostrará cómo se preparan los programas. Se puede escribir más de una instrucción en una sola sentencia REM, siempre que las instrucciones vayan separadas por un punto y coma (;).

El ensamblador reconoce todas las instrucciones del Z80 que se muestran en el Apéndice A y también algunos directivos, que se listan más adelante. Los directivos son instrucciones que no se traducen a instrucciones en código máquina pero se utilizan para dar indicaciones al programa ensamblador. A los directivos se les llama con frecuencia pseudooperaciones porque se asemejan a las instrucciones que se traducen a código máquina.

A lo largo del libro todas las instrucciones que van incluidas en el texto se han impreso en letras mayúsculas para que se puedan distinguir fácilmente del resto del texto. Cuando se utilice el Ensamblador del ZX Spectrum se tienen que introducir todas las instrucciones en letras *minúsculas*, como se indica en la contraportada del manual del Spectrum.

Los números utilizados con el Ensamblador del ZX Spectrum pueden ser decimales y hexadecimales. Los números hexadecimales van precedidos por un signo dólar (por ejemplo, $1AB5). Se pueden utilizar rótulos para referirse a posiciones de memoria y el ensamblador convertirá automáticamente el rótulo a una dirección correcta de memoria. Los rótulos en el Ensamblador ZX Spectrum pueden ser de cualquier longitud, pero el primer carácter tiene que ser una letra mayúscula. La otra única restricción que tienen los rótulos es que no deben contener los caracteres «)» o «+». Cuando se utilice al comienzo de una instrucción como puntero a esa instrucción (o directivo), se trata como una instrucción independiente y va seguida de un punto y coma.

Existen tres tipos de mensajes de error que pueden producirse durante el ensamblado. Si hay algún error en los directivos (pseudoinstrucciones) GO, FINISH u ORG se producirá un mensaje de error antes de que comience el ensamblado. Una instrucción incorrecta se mostrará mediante un mensaje parpadeante que indica el número de línea, el número de la instrucción dentro de la línea, el tipo de instrucción. Esto debe permitirle encontrar fácilmente la instrucción que contiene el error; por desgracia no siempre es tan sencillo encontrar el error en sí. Finalmente, puede obtener uno de los mensajes de error de Sinclair. Los mensajes posibles son:

B Entero fuera de rango—un número fuera del rango permitido.

2 Variable no encontrada—referencia a un rótulo inexistente.

Q Parámetro erróneo—instrucción erróneamente tecleada.

6 Número demasiado grande—desplazamiento de un salto relativo fuera de rango.

B.3 Directivos (pseudoinstrucciones)

go—todos los programas en Ensamblador del ZX Spectrum tienen que comenzar por esta instrucción. Tiene que estar en su propia sentencia REM.

finish—ésta tiene que ser la última instrucción de todo programa. De nuevo tiene que ir en su propia sentencia REM.

org—esta instrucción le indica al ensamblador qué posición de memoria ha de utilizar para el comienzo del programa en código máquina. En un Spectrum estándar de 16K o 48K, org 23 760 cargará el programa en código máquina en el primer byte libre de una sentencia REM al comienzo del programa. **org** debería ser siempre la segunda instrucción del programa en lenguaje ensamblador, pero puede utilizarse en medio de un programa para hacer que el programa salte a una nueva posición de memoria.

defb—permite que uno o más bytes de memoria, comenzando en la dirección actual de ensamblaje, se pongan a un valor definido en el rango 0 a 255.

defw—permite que una palabra (dos bytes) de memoria se ponga un valor comprendido en el rango 0 a 65 535.

defs—permite colocar una cadena de caracteres en memoria.

equ—utilizado para asignar un rótulo a una posición de memoria. Muy útil cuando la dirección de esa posición está fuera del programa en código máquina. Puede utilizarse para encadenar secciones de código máquina.

APENDICE C

TABLAS DE CONVERSION HEXADECIMAL-DECIMAL

Este libro se ha escrito utilizando principalmente números decimales, puesto que generalmente son más sencillos de comprender para las personas. Sin embargo, hay ocasiones en las que la utilización de números binarios o hexadecimales tienen un mayor significado, en particular cuando se trata de patrones de bits en un registro o en una posición de memoria de 16 bits a dos números de ocho bits. Las siguientes tablas proporcionan la conversión entre hexadecimal y decimal. La tabla C.1 proporciona la conversión para números hexadecimales hasta FF o 255 en decimal y la tabla C.2, junto con esta primera, entre todos los números hasta el FFFF o 65 535 en decimal.

Tabla C.1
Conversión de números hexadecimales hasta el FF o 255 en decimal

	0	1	2	3	4	5	6	7	8	9	A	B	C	B	C	D
00	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
10	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
20	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
30	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
40	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
50	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
60	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
70	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
80	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
90	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
AO	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
BO	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
CO	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
DO	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
EO	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
FO	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

Tabla C.2
Conversión de números hexadecimales hasta el FFFF o 65 535 en decimal (junto con la tecla C.1)

Hexadecimal	Decimal	Hexadecimal	Decimal
100	256	1000	4096
200	512	2000	8192
300	768	3000	12288
400	1024	4000	16384
500	1280	5000	20480
600	1536	6000	24576
700	1792	7000	28672
800	2048	8000	32768
900	2304	9000	36864
A00	2560	A000	40960
B00	2816	B000	45056
C00	3072	C000	49152
D00	3328	D000	53248
E00	3584	E000	57344
F00	3840	F000	61440

APENDICE D
ENSAMBLAJE MANUAL

D.1 Método general

Un programa escrito en lenguaje ensamblador tiene que traducirse a código máquina antes de poderlo ejecutar en una computadora. La forma más sencilla de hacerlo es utilizar un programa ensamblador. Para aquellos que están interesados en escribir programas que no serán demasiado pequeños, se recomienda encarecidamente la utilización de un ensamblador. El otro método de traducir a código máquina es utilizar tablas de instrucciones y convertir el programa de forma manual.

Para traducir de lenguaje ensamblador a código máquina, tiene que consultarse cada instrucción en tablas de instrucciones, como, por ejemplo, los del Apéndice A, para buscar la forma numérica de la instrucción. Este número puede ser binario, hexadecimal o decimal; si el número está binario, generalmente se convierte a hexadecimal, que es más sencillo, o a decimal.

Después de traducir el programa a una lista de números se tiene que cargar en la memoria de la computadora. A los números decimales se les puede hacer POKE a memoria mediante un programa muy sencillo. Los números hexadecimales tienen que convertirse previamente a decimal para que les pueda hacer POKE a memoria. Por supuesto, antes de introducir el programa en memoria tiene que decidir en qué parte de la memoria irá y quizá tenga que introducir instrucciones para reservar un área de memoria para el programa.

D.2 Direcciones y datos

Además de convertir todas las instrucciones a su forma numérica, tiene también que convertir todos sus datos a números en la misma base que las instrucciones. La computadora sólo distingue entre datos e instrucciones, conociendo qué debería haber en la siguiente posición de memoria. Por ejemplo, si la dirección de comienzo de un programa es la 32 000, la computadora tomará el número que encuentre en la posición de memoria 32 000 como una instrucción. Si el número de la posición 32 000 se traduce en una instrucción que debe ir seguida por datos como, por ejemplo, la instrucción LD A,n, entonces la computadora tomará el número de la posición como el valor de n.

Uno de los mayores problemas que tiene el ensamblaje manual es la traducción de números de 16 bits. Estos se deben introducir en la computadora como dos números de ocho bits, introduciendo primero los ocho bits de la derecha antes que los ocho bits de la izquierda. La figura D.1 muestra la conversión del número decimal 32 000 en dos números de ocho bits que se convierten a decimal. Mediante este ejemplo la instrucción CALL 32 000 se colocaría en tres posiciones de memoria como 205, 0, 125.

```
32000 = 0111110100000000B
      = 01111101/00000000
      = 125      0
32000 = 0, 125
```

Figura D.1

D.3 Instrucciones de salto

Hay que considerar dos tipos de instrucciones de salto: saltos absolutos (instrucciones JP) y saltos relativos (instrucciones JR). El método de traducción es el mismo para los saltos condicionales e incondicionales.

Hay saltos absolutos que van seguidos por la dirección real de la posición de memoria que contiene la instrucción siguiente al salto. Si está saltando hacia adelante del programa tendrá que esperar hasta que haya traducido el programa hasta esa instrucción para saber la dirección adecuada. Recuerde que los bytes de las direcciones se tendrán que invertir, como mostrábamos en la sección anterior.

Los saltos relativos causan la mayoría de los problemas cuando se ensamblan los programas manualmente. Si su programa no funciona correctamente lo mejor será que vuelva a calcular sus saltos relativos. El dato de una instrucción de salto relativo es el número de posiciones de memoria desde la instrucción de salto hasta la siguiente instrucción menos dos. Si es un salto hacia atrás en el programa, será un número negativo de posiciones que se expresa como un número de ocho bits en «complemento a dos». Se restan dos del desplazamiento porque cuando se ejecuta la instrucción el contador del programa está apuntando ya a la siguiente instrucción. El programa de la figura D.2 muestra la traducción de dos saltos relativos.

Lenguaje ensamblador		Código máquina	Comentario
Bucle;	1d a,(hl)	126	
	cp0	254	
		0	
	jr z,Final	40	Saltar cinco posiciones hacia delante
		3	(5–2)
	inc hl	35	
	jr bucle	24	Saltar seis posiciones hacia atrás
		248	(–6–2 en complemento a dos)
Final;	ret	201	

Figura D.2

D.4 Instrucciones de bit

Puesto que las instrucciones de comprobación, establecimiento o borrado de bits en un registro dependen del registro y del número de bit, se tiene que poner mucha atención para asegurar que se utiliza el código de instrucción correcto. Todas las instrucciones de bit tienen el código 203 en el primer byte de la instrucción.

D.5 Registros índice

Las instrucciones con registros índice parecen más complejas que la mayoría de las instrucciones porque muchas de ellas son instrucciones de tres o cuatro bytes. Pueden de hecho simplificarse de forma apreciable si observamos que tienen el mismo código máquina que las instrucciones equivalentes para la pareja de registros HL, excepto que las instrucciones que utilizan el registro IX van precedidas por el byte 221 y las instrucciones del registro IY van precedidas por el byte 253.

Ambos registros, cuando se utilizan como apuntadores de memoria, tienen que incluir un byte que muestre el desplazamiento desde la posición contenida en el registro índice, incluso aunque éste sea cero.

APENDICE E
CODIGOS DE CARACTER

La tabla de este apéndice, tabla E.1, lista los caracteres y sus códigos que pueden introducirse desde el teclado. A parte de los caracteres gráficos son también los caracteres que más se utilizan en las salidas.

Tabla E.1

32	(espacio)	80	P
33	!	81	Q
34	"	82	R
35	≠	83	S
36	$	84	T
37	%	85	U
38	&	86	V
39	'	87	W
40	(	88	X
41	)	89	Y
42	*	90	Z
43	+	91	[
44	,	92	/
45	–	93	]
46	.	94	↑
47	/	95	—
48	0	96	£
49	1	97	a
50	2	98	b
51	3	99	c
52	4	100	d
53	5	101	e
54	6	102	f
55	7	103	g
56	8	104	h
57	9	105	i
58	:	106	j
59	;	107	k
60	<	108	l

Tabla E.1 (continuación)

61	=	109	m
62	>	110	n
63	?	111	o
64	@	112	p
65	A	113	q
66	B	114	r
67	C	115	s
68	D	116	t
69	E	117	u
70	F	118	v
71	G	119	w
72	H	120	x
73	I	121	y
74	J	122	z
75	K	123	{
76	L	124	|
77	M	125	}
78	N	126	~
79	O	127	©

APENDICE F
CARACTERES PARA EL CONTROL DE LA IMPRESION

Código	Efecto
6	Imprimir coma (mover media pantalla)
8	Espacio hacia atrás
13	Próxima línea
16	INK
17	PAPER
18	FLASH
19	BRIGHT
20	INVERSE
21	OVER
22	AT
23	TAB

APÉNDICE G
SUBRUTINAS ROM

G.1 El programa de la ROM

Cuando enciende su Spectrum, un programa comienza a ejecutarse inmediatamente; este es el programa en ROM (Memoria sólo de lectura) y ocupa las primeras 16K de la memoria disponible. El propósito de este programa es permitir al microprocesador que se comunique con varios dispositivos de entrada y salida utilizados por el Spectrum y la introducción y ejecución de programas BASIC. En el sistema Spectrum estándar, las entradas principales vienen desde el teclado o desde la casete y las salidas se envían a la pantalla del televisor, al altavoz, a la grabadora de casetes y la impresora.

El programa ROM es un programa en código máquina y está escrito en forma de subrutinas. Esto significa que estas subrutinas están disponibles para sus programas en lenguaje ensamblador. Este apéndice listará algunas rutinas de las más sencillas y les mostrará cómo utilizarlas.

G.2 Imprimir un carácter

El carácter que se encuentra en el registro A puede utilizarse en el canal actualmente seleccionado mediante la sola instrucción:

```
RST 16
```

Para imprimir en la parte superior de la pantalla, se tiene primero que abrir el canal adecuado y después el siguiente segmento de programa imprimirá un carácter en la parte superior de la pantalla:

```
LD A,2
CALL 5633
LD A,Código; !carácter a imprimir
RST 16
```

Como hemos visto anteriormente, esta rutina puede utilizarse también para modificar la posición actual de impresión y los colores temporales mediante los códigos de control de la impresión dados en el Apéndice F.

G.3 Borrado de la pantalla

Se puede borrar toda la pantalla mediante el siguiente segmento de programa:

```
LD A,2
CALL 5633
CALL 3435
```

Hay otra rutina que se puede utilizar para borrar solamente parte de la pantalla. Esta rutina borra un número determinado de líneas, contando desde la parte inferior de la pantalla:

```
LD B,Líneas; !Número de líneas a borrar
CALL 3652
```

G.4 Scroll (Desplazamiento vertical o enrollamiento) de la pantalla

A la pantalla se le puede hacer que haga *scroll* (desplazamiento vertical) de forma automática cargando de forma repetitiva un valor superior a uno en la variable del sistema SCR CT. Probablemente la mejor elección es 255. Utilice las instrucciones:

```
PUSH HL
LD HL,23692
LD (HL),255
POP HL
```

Hay una rutina en la ROM que permite establecer el número de líneas a los que se le harán scroll. De nuevo el número de líneas se cuenta desde la parte inferior de la pantalla. El valor encargado en el registro B será uno menos del número de líneas a los que se hará scroll:

```
LD B,Líneas
CALL 3584
```

G.5 Color del margen

El color del margen se modifica colocando el número del color requerido en el registro A y después llamando a una subrutina:

```
LD A,Color
CALL 8859
```

G.6 Colores de la pantalla

Las variables que indican los colores se almacenan como bytes en el fichero de atributos y en las variables del Spectrum ATTR-P, ATTR-T, MASK-P y MASK-T. Generalmente las rutinas de la ROM del Spectrum utilizan valores temporales de color, pero algunas, como la rutina de borrado de pantalla, utilizan los valores permanentes.

Los atributos permanentes se pueden establecer modificando los bits adecuados de la variable del sistema ATTR-P; ella está en la posición 23 693. Los atributos se almacenan de la siguiente forma:

Bits 0-2	Color de INK (tinta).
Bits 3-5	Color de PAPER (papel).
Bit 6	Establecimiento del BRIGHT (brillo).
Bit 7	Establecimiento del FLASH (parpadeo).

Una vez establecido los colores, bien como permanentes o como temporales, se pueden utilizar las siguientes rutinas. Para copiar los valores permanentes a las variables temporales del sistema utilice la instrucción:

CALL 3405

y para copiar los valores temporales a los permanentes utilice:

CALL 7341

G.7 Entrada desde el teclado

La rutina más sencilla de utilizar es aquella que verifica si se ha pulsado una tecla del teclado. Si se ha pulsado se coloca un valor en la pareja de registros DE. Esta rutina se puede utilizar con la instrucción:

CALL 654

La rutina principal para la entrada de caracteres se dio en el Capítulo 6.

G.8 Sonido

La rutina que envía una sola nota al altavoz se puede llamar con la instrucción:

CALL 949

Antes de utilizar la instrucción hay que cargar el registro HL con un valor que establece el tono de la nota y el registro DE hay que cargarlo con un valor que determina la longitud de la nota.

G.9 La impresora

Hay dos rutinas en la ROM que pueden utilizarse con la impresora. La más sencilla es la rutina COPY que copia la pantalla a la impresora. Se utiliza mediante la instrucción:

CALL 3691

El contenido de la memoria auxiliar de la impresora se pasa a la impresora mediante la instrucción:

CALL 3789

G.10 Gráficos

Las rutinas de gráficos para hacer PLOT y DRAW se pueden llamar desde la ROM para proporcionar fácilmente gráficos de alta resolución. La rutina PLOT requiere que se cargue la posición x en el registro C y la posición y en el registro B antes de llamar a la subrutina con:

CALL 8927

La rutina DRAW es más complicada porque se pueden utilizar valores positivos y negativos para x e y. Antes de llamar a la subrutina los registros B y C deberían contener los valores absolutos de y y x respectivamente y los registros D y E deberían contener los signos de x e y respectivamente. Si x es positivo, D contendrá el valor 1; si es negativo, D contendrá – 1; y finalmente si x es cero, D contendrá el valor cero. La instrucción para llamar a la rutina es:

CALL 9402

INDICE